Django CRUD App

Internship Project

BLACKBUCKS
GROUP OF COMPANIES

Abstract

The Django CRUD App project showcases the implementation of basic CRUD (Create, Read, Update, Delete) operations using the Django web framework. The primary objective of the project is to provide a simple, yet effective solution for managing employee records in a web application. Utilizing SQLite3 for database management and Bootstrap for responsive design, the project emphasizes ease of use and efficiency.

The application is built on Django's Model-View-Template (MVT) architecture, ensuring a clean separation of concerns and streamlined development process. Key functionalities include a home page listing all records, forms for creating and updating records, detailed view pages, and delete functionality. The project demonstrates effective integration of front-end and back-end technologies, resulting in a robust and user-friendly application.

The successful implementation of this Django CRUD App serves as a practical learning tool for developers, highlighting the advantages of using Django for web development. Future enhancements could include user authentication and more complex data interactions, further extending the application's capabilities.

Technology Stack Introduction and Theoretical Background

Django

Introduction: Django is a high-level Python web framework that promotes rapid development and clean, pragmatic design. Built by experienced developers, it handles much of the complexity of web development, allowing you to focus on writing your app without needing to reinvent the wheel.

Key Features:

- **Fast and Secure:** Django helps developers avoid many common security mistakes and makes it easier to produce a secure website.
- **Scalable:** Django uses a component-based "shared-nothing" architecture. Each part of the architecture is independent and can be replaced or changed as needed.
- **Versatile:** Django can be (and has been) used to build almost any type of website – from content management systems and wikis to social networks and news sites.

Theoretical Background: Django follows the Model-View-Template (MVT) architectural pattern, which is similar to the Model-View-Controller (MVC) pattern. In Django, the "Controller" is handled by the framework itself, leaving the developer to work on the model, view, and template.

- **Model:** Represents the data structure. It defines the database schema and business logic.
- **View:** Manages the user interface logic and processes user requests.
- **Template:** Deals with the presentation layer, rendering HTML to be sent to the client.

SQLite3

Introduction: SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. It is the most widely deployed database engine in the world.

Key Features:

- **Serverless:** SQLite is serverless, which means it doesn't require a separate server process or system to operate.
- **Zero-Configuration:** No setup or administration required.
- **Cross-Platform:** Runs on any platform, including macOS, Windows, Linux, etc.

Theoretical Background: SQLite is an embedded SQL database engine, unlike most other SQL databases which are implemented as a separate server. Applications interact with SQLite through function calls rather than sending messages to a server process. The database is stored in a single file on disk, making it a convenient choice for applications needing a lightweight database solution.

HTML

Introduction: HTML (HyperText Markup Language) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as CSS and JavaScript.

Key Features:

- **Semantic Structure:** HTML elements form the building blocks of all websites, providing the structure for web pages.
- **Compatibility:** Supported by all browsers, ensuring consistent display of web pages across different devices and platforms.

Theoretical Background: HTML uses tags to structure content. Elements like headings, paragraphs, links, images, and forms are all defined using HTML tags. HTML5 introduced new elements and attributes, providing more functionality and a better way to structure web content.

Bootstrap

Introduction: Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

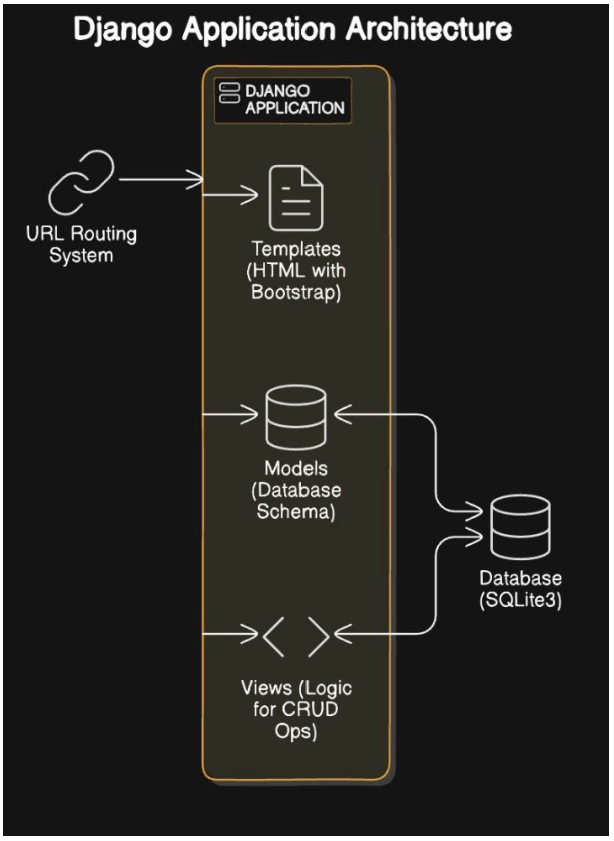
Key Features:

- **Responsive Design:** Ensures web pages adjust seamlessly to various screen sizes.
- **Pre-designed Components:** Offers a variety of reusable components like modals, dropdowns, alerts, etc.
- **Customizable:** Easily customizable to fit the needs of any project.

Theoretical Background: Bootstrap uses a 12-column grid system for layout management, making it simple to create responsive designs. It also includes a wide array of pre-styled components, which can be customized using CSS variables. Bootstrap's utility classes allow for quick styling adjustments without writing custom CSS.

Architecture

The architecture of the Django CRUD application follows the Model-View-Template (MVT) pattern, which is similar to the Model-View-Controller (MVC) pattern used in other frameworks. This architecture ensures a clear separation of concerns and allows for efficient



development and maintenance of the application.

Components of MVT Architecture:

1. **Model:**
    - Represents the data and the business logic.
    - Defines the structure of the database, including the fields and their data types.
    - Interacts with the database through Django's Object-Relational Mapping (ORM) to perform CRUD operations.
2. **View:**
    - Handles the user interface logic and processes user requests.
    - Retrieves data from the model and sends it to the template for rendering.
    - Handles form submissions and user input validation.
3. **Template:**
    - Manages the presentation layer and renders HTML to be sent to the client.

- Uses Django's templating language to dynamically generate HTML content based on the data provided by the view.
- Ensures a clean separation of presentation logic from business logic.

Example Flow of a Request in MVT:

1. **User Request:**
   - A user sends a request to the server by accessing a URL (e.g., http://example.com/employees/).
2. **URL Dispatcher:**
   - Django's URL dispatcher maps the requested URL to the appropriate view function based on the URL patterns defined in `urls.py`.
3. **View Processing:**
   - The view function processes the request, interacting with the model to retrieve or modify data.
   - The view prepares the data and passes it to the template for rendering.
4. **Template Rendering:**
   - The template generates the HTML content using the data provided by the view.
   - The rendered HTML is sent back to the client as the response.
5. **Client Display:**
   - The user's browser receives the HTML response and displays the content to the user.

Design or Flow of the Project

The design and flow of the Django CRUD application are structured to provide a user-friendly interface for managing employee records through Create, Read, Update, and Delete operations. The application consists of several key pages and functionalities, each serving a specific purpose.

Key Pages and Their Functions:

1. **Home Page:**
    - Lists all employee records stored in the database.
    - Provides navigation options to create a new record, view details of existing records, update records, and delete records.
    - Displays a table with columns such as Employee ID, Name, Email, Address, and Phone Number.
2. **Create Page:**
    - Presents a form to add a new employee record to the database.
    - Includes fields for Employee ID, Name, Email, Address, and Phone Number.
    - Validates the input data and saves the new record to the database upon form submission.
3. **Read/View Page:**
    - Displays detailed information about a selected employee record.
    - Retrieves data from the database based on the employee ID.
    - Provides a user-friendly view of the employee's details.
4. **Update Page:**
    - Presents a form pre-filled with the current data of the selected employee record.
    - Allows the user to edit the details and save the changes.
    - Validates the updated data and saves the changes to the database upon form submission.
5. **Delete Functionality:**
    - Provides an option to delete an employee record from the database.
    - Prompts the user for confirmation before proceeding with the deletion.
    - Removes the record from the database upon confirmation.

Flow of Operations:

1. **Home Page (List View):**
    - User visits the home page to see a list of all employee records.
    - User can navigate to the Create, Read, Update, or Delete functionalities from the home page.
2. **Create Operation:**
    - User clicks on the "Add New Employee" button/link.

- User is redirected to the Create Page with a form.
- User fills out the form and submits it.
- Application validates the data and saves the new record to the database.
- User is redirected back to the home page with the updated list of records.

3. **Read Operation:**
   - User clicks on the "View Details" link for a specific employee.
   - User is redirected to the Read/View Page with detailed information about the selected employee.
   - User can navigate back to the home page or other pages.
4. **Update Operation:**
   - User clicks on the "Edit" link for a specific employee.
   - User is redirected to the Update Page with a pre-filled form.
   - User edits the details and submits the form.
   - Application validates the updated data and saves the changes to the database.
   - User is redirected back to the home page with the updated list of records.
5. **Delete Operation:**
   - User clicks on the "Delete" link for a specific employee.
   - Application prompts the user for confirmation.
   - Upon confirmation, the application deletes the record from the database.
   - User is redirected back to the home page with the updated list of records

Detailed Description of the Database Schema and ER Diagram

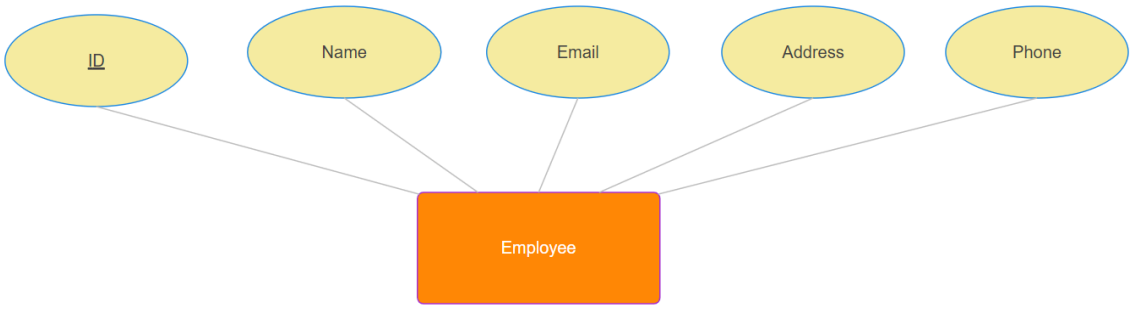
The database schema for the Django CRUD application is designed to efficiently store and manage employee information. This schema includes fields such as Employee ID, Name, Email, Address, and Phone Number. The simplicity of the schema allows for effective demonstration and execution of CRUD operations within the application.

Database Schema

1. **Employee Table:**
    - **Employee ID:** Integer, primary key, unique identifier for each employee.
    - **Name:** String, stores the full name of the employee.
    - **Email:** String, stores the email address of the employee.
    - **Address:** String, stores the residential address of the employee.
    - **Phone Number:** String, stores the contact phone number of the employee.

Entity-Relationship (ER) Diagram

The ER diagram for the database schema is straightforward, representing the single entity "Employee" with its attributes.



Explanations of SQL Code and Queries

The application utilizes Django's Object-Relational Mapping (ORM) to interact with the SQLite3 database. The following examples demonstrate how Django ORM translates into SQL queries to perform various CRUD operations.

Create: Adding a New Employee Record

Django ORM:

```
Employee.objects.create(employee_id=1, name="Dhanunjay",
email="jagudhanunjai@gmail.com", address="Kkd",
phone="807578685")
```

Equivalent SQL:

```
INSERT INTO employee (employee_id, name, email, address,
phone) VALUES (1, 'Dhanunjay', 'jagudhanunjai@gmail.com',
'Kkd', '807578685');
```

Read: Retrieving All Employee Records

Django ORM:

```
Employee.objects.all()
```

Equivalent SQL:

```
SELECT * FROM employee;
```

Update: Updating an Employee's Information

Django ORM:

```
employee = Employee.objects.get(id=1)

employee.name = "Dhanunjay Updated"

employee.save()
```

Equivalent SQL:

```
UPDATE employee SET name = 'Dhanunjay Updated' WHERE id = 1;
```

Delete: Deleting an Employee Record

Django ORM:

```
employee = Employee.objects.get(id=1)

employee.delete()
```

Equivalent SQL:

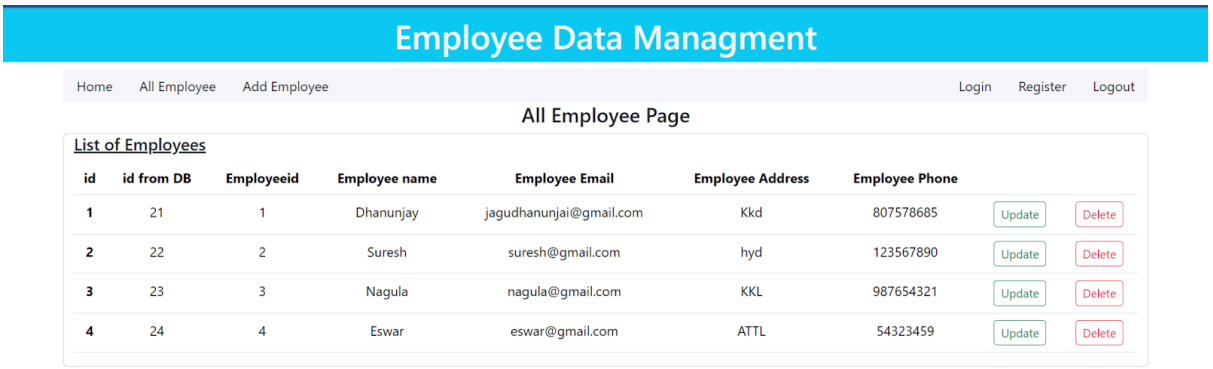
```
DELETE FROM employee WHERE id = 1;
```

Database and Query Results

The following screenshot shows the "All Employee Page," which displays all employee records stored in the database. This page lists each employee's ID, Name, Email, Address, and Phone Number.

Screenshot: All Employee Page

(A screenshot here showing the list of employees in the application.)



Requests to the Database in Terminal

During the development and debugging phases, you can observe the requests made to the database through the terminal. These requests demonstrate the interaction between the Django application and the SQLite3 database.

(A screenshot or a snippet of the terminal output showing the database requests.)

```
[12/Jul/2024 01:11:12] "GET /deleteemployee/19/ HTTP/1.1" 404 3236
[12/Jul/2024 01:12:25] "GET /allemployees/ HTTP/1.1" 200 3226
[12/Jul/2024 01:12:25] "GET /allemployees/ HTTP/1.1" 200 3226
[12/Jul/2024 01:12:26] "GET /allemployees/ HTTP/1.1" 200 3226
[12/Jul/2024 01:12:37] "GET /allemployees/ HTTP/1.1" 200 3217
[12/Jul/2024 01:12:58] "GET /allemployees/ HTTP/1.1" 200 3235
[12/Jul/2024 01:12:58] "GET /allemployees/ HTTP/1.1" 200 3235
[12/Jul/2024 01:13:13] "GET /deleteemployee/18/ HTTP/1.1" 302 0
[12/Jul/2024 01:13:13] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:13:14] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:13:15] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:13:16] "GET /addemployee/ HTTP/1.1" 200 3527
[12/Jul/2024 01:13:21] "POST /addemployee/ HTTP/1.1" 302 0
[12/Jul/2024 01:13:21] "GET /allemployees/ HTTP/1.1" 200 3239
[12/Jul/2024 01:13:25] "GET /deleteemployee/20/ HTTP/1.1" 302 0
[12/Jul/2024 01:13:25] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:14:43] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:14:44] "GET /allemployees/ HTTP/1.1" 200 2684
[12/Jul/2024 01:14:45] "GET /addemployee/ HTTP/1.1" 200 3527
[12/Jul/2024 01:14:50] "POST /addemployee/ HTTP/1.1" 302 0
[12/Jul/2024 01:14:50] "GET /allemployees/ HTTP/1.1" 200 3110
[12/Jul/2024 01:15:04] "GET /allemployees/ HTTP/1.1" 200 3143
[12/Jul/2024 01:15:06] "GET /allemployees/ HTTP/1.1" 200 3143
[12/Jul/2024 01:15:14] "GET /updateemployee/21/ HTTP/1.1" 200 3611
[12/Jul/2024 01:15:19] "POST /doupdateemployee/21/ HTTP/1.1" 302 0
[12/Jul/2024 01:15:19] "GET /allemployees/ HTTP/1.1" 200 3152
[12/Jul/2024 01:15:20] "GET /updateemployee/21/ HTTP/1.1" 200 3620
[12/Jul/2024 01:15:22] "POST /doupdateemployee/21/ HTTP/1.1" 302 0
[12/Jul/2024 01:15:22] "GET /allemployees/ HTTP/1.1" 200 3166
[12/Jul/2024 01:15:34] "GET /updateemployee/21/ HTTP/1.1" 200 3634
[12/Jul/2024 01:16:02] "POST /doupdateemployee/21/ HTTP/1.1" 302 0
[12/Jul/2024 01:16:02] "GET /allemployees/ HTTP/1.1" 200 3168
```

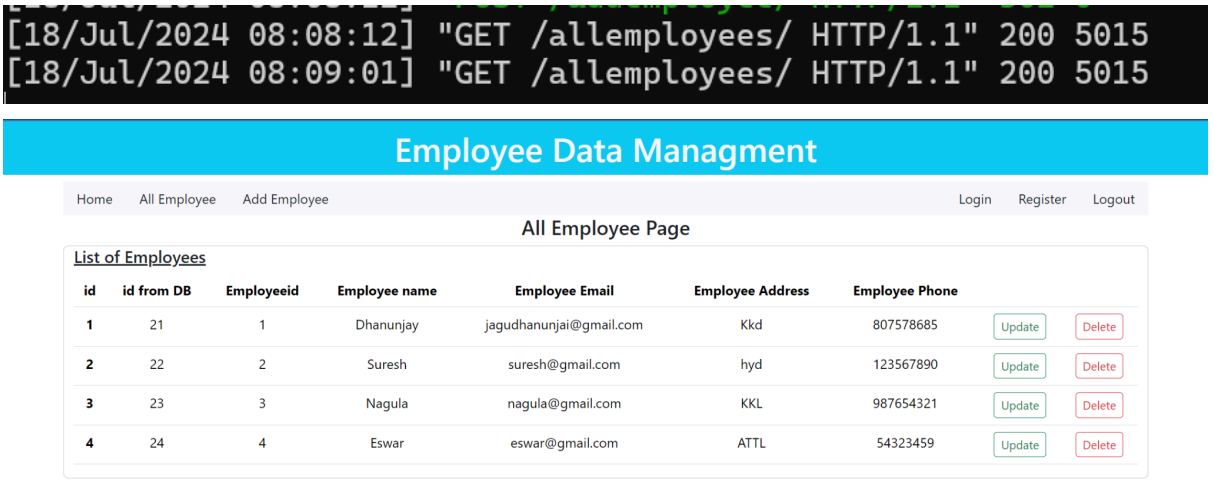
Working of the Application

Home Page

The home page lists all records with options to create, read, update, or delete entries. It acts as the central hub from which users can navigate to other parts of the application.

Flow:

1. **List Records:** Fetches all records from the database and displays them in a table format.



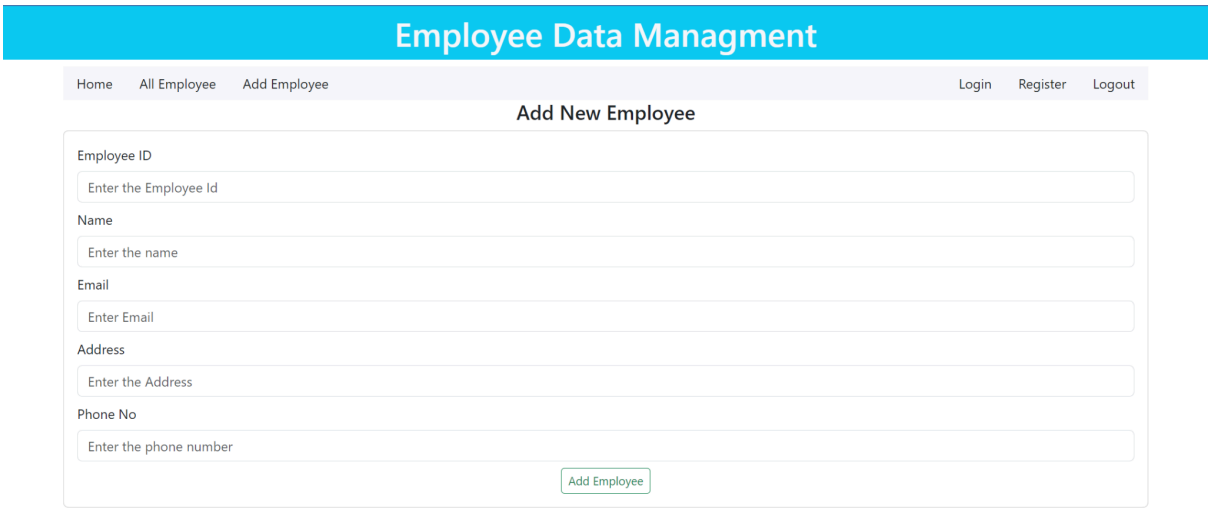
2. **Navigation Links:** Provides links/buttons for creating a new record, viewing detailed information, editing, and deleting existing records.

Create Page

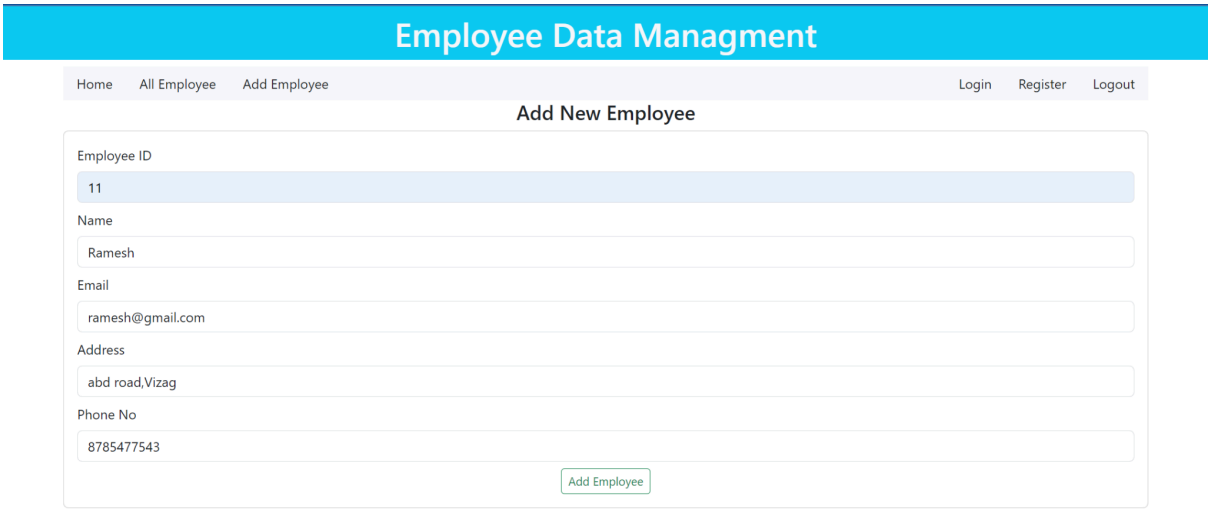
The create page provides a form to add new records to the database. It includes fields such as Employee ID, Name, Email, Address, and Phone Number. Upon submission, the form data is validated and saved to the database.

Flow:

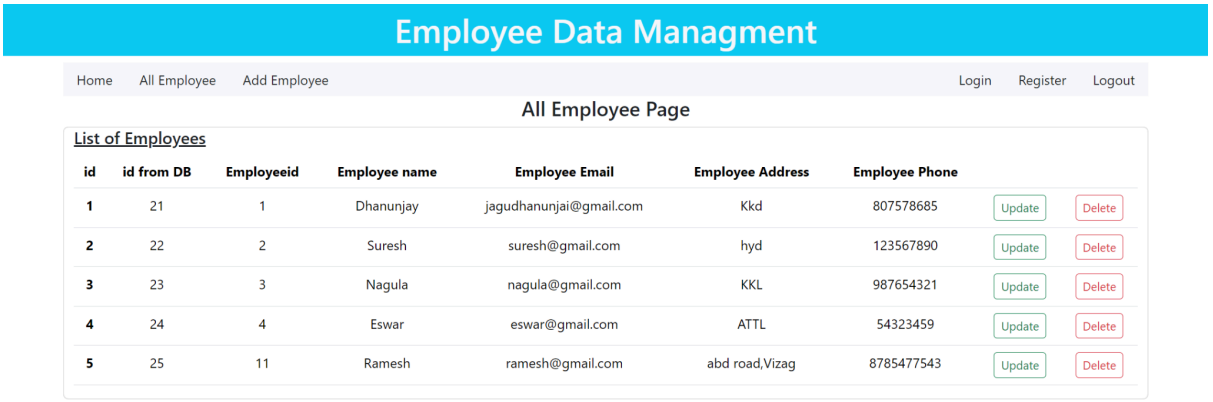
1. **Form Display:** Renders a form with fields for each required piece of information.



2. **Form Submission:** Handles the submission of the form, validates the input, and saves the data to the database.



3. **Redirection:** Redirects back to the home page upon successful creation of the record.

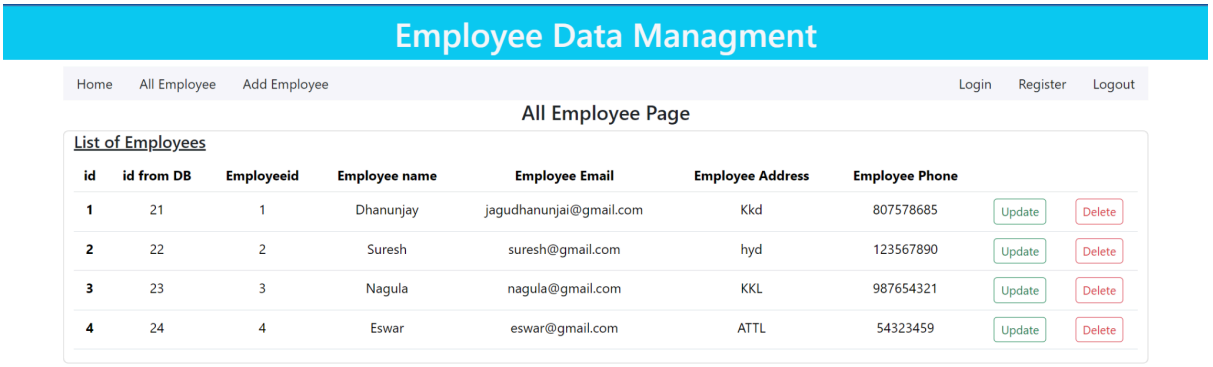


Read/View Page

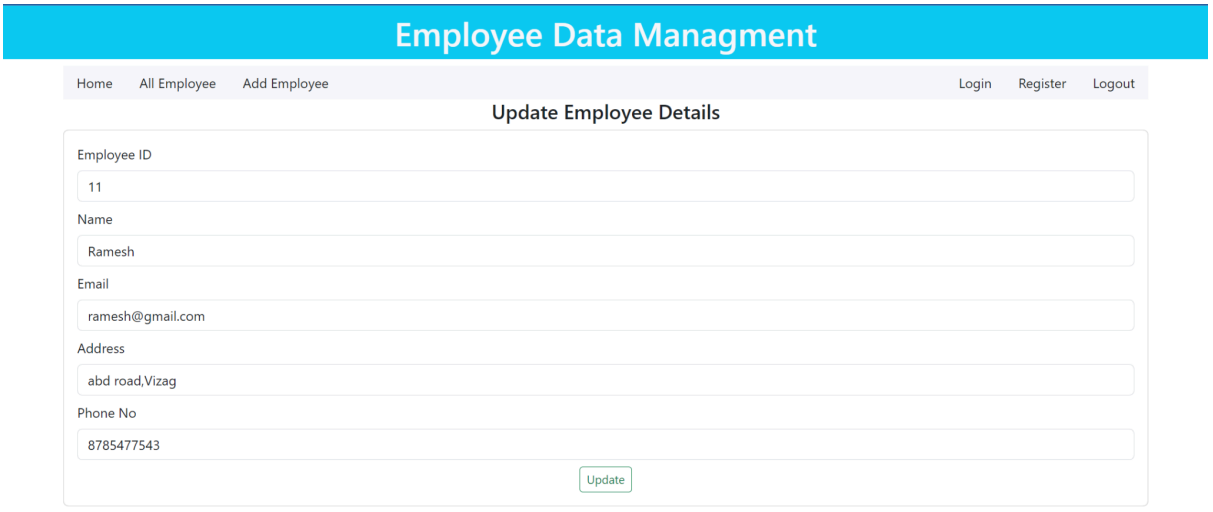
The read/view page displays details of a selected record. It retrieves data from the database and presents it in a user-friendly format.

Flow:

1. **Fetch Data:** Retrieves the specific record from the database using its ID.



2. **Display Details:** Renders the detailed information of the selected record.

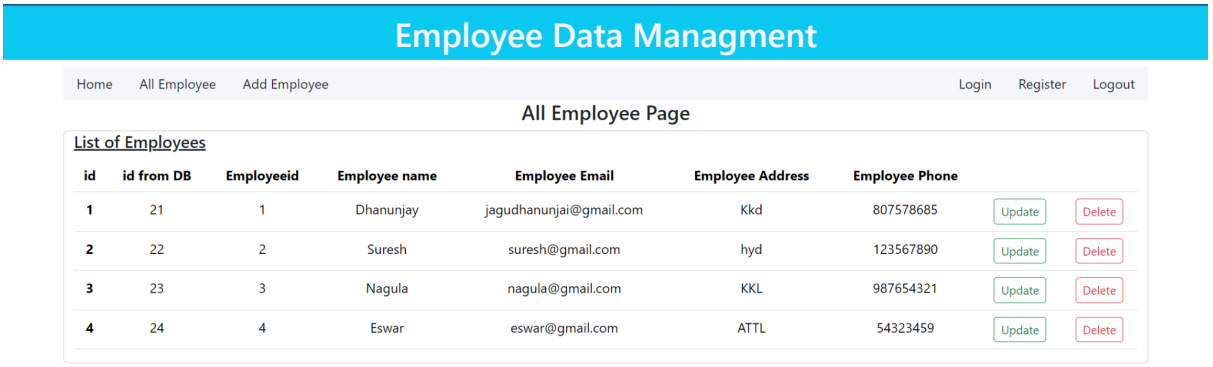


Update Page

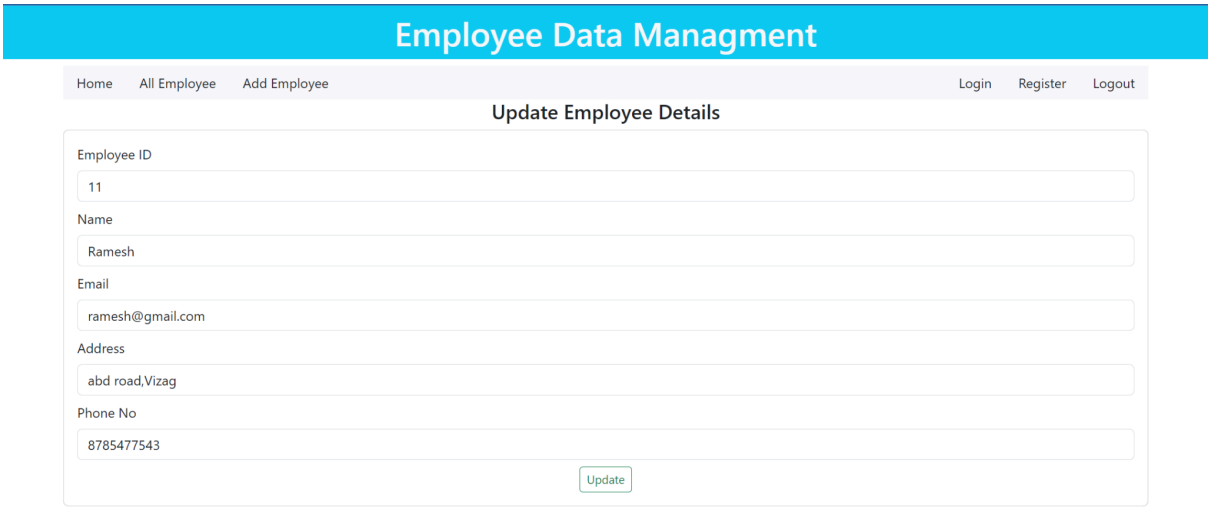
The update page allows editing of an existing record. It pre-fills the form with the current data, enabling users to make changes and save the updates to the database.

Flow:

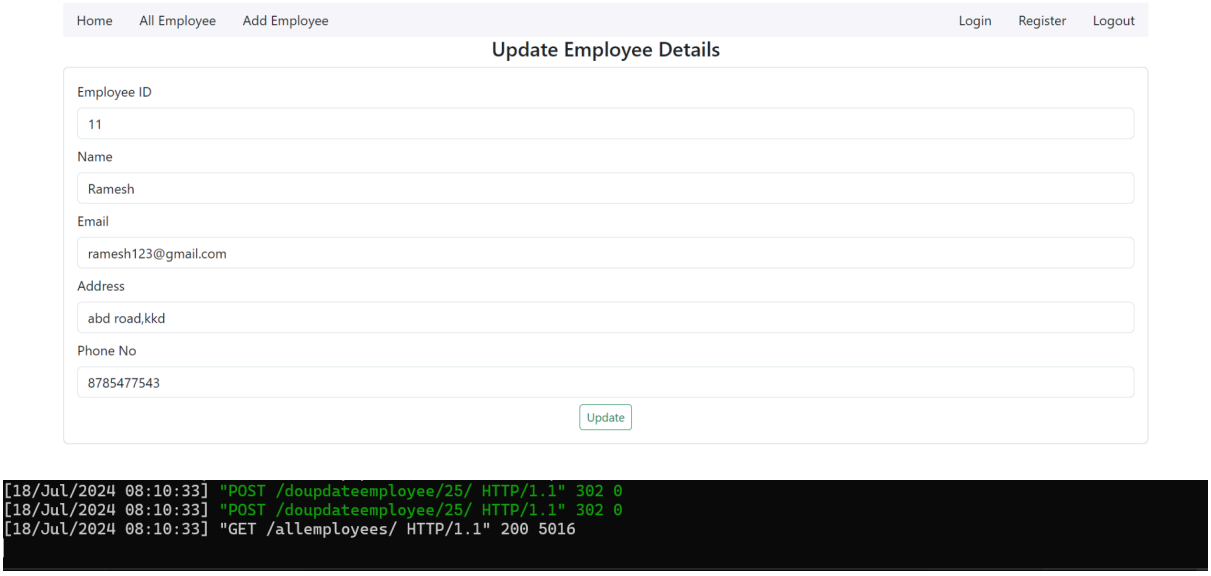
1. **Fetch Data:** Retrieves the specific record to be updated from the database.



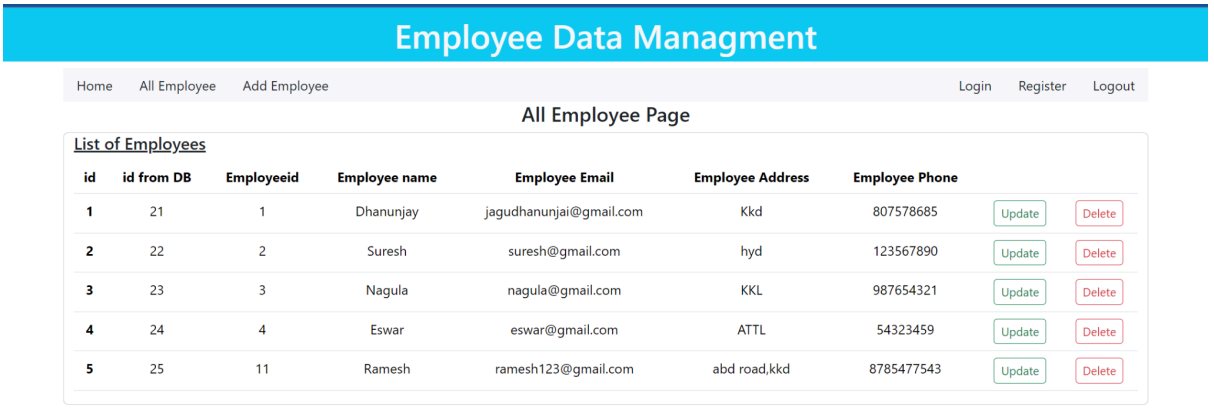
2. **Form Display:** Renders the form pre-filled with the current data.

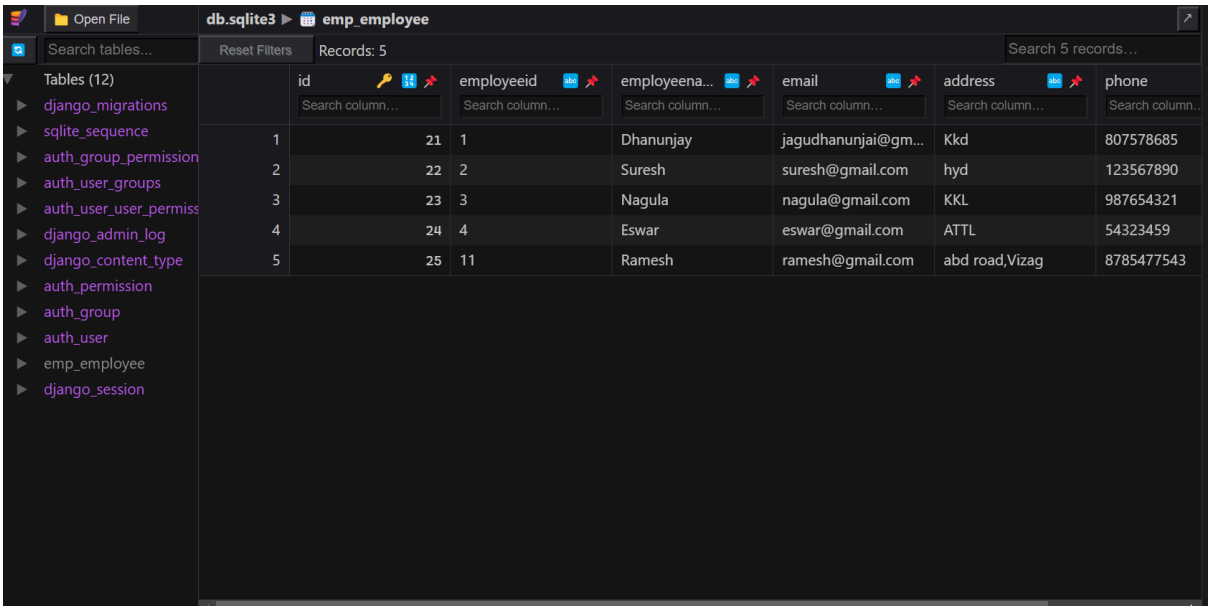


3. **Form Submission:** Handles the submission of the form, validates the input, and updates the record in the database.



4. **Redirection:** Redirects back to the home page upon successful update of the record.



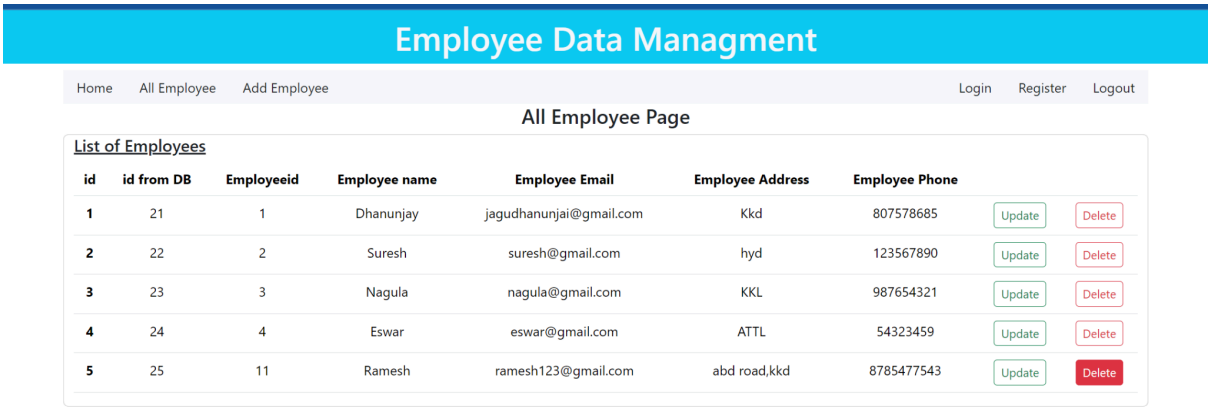


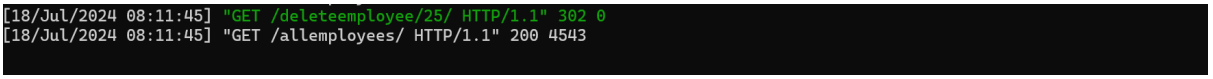
Delete Functionality

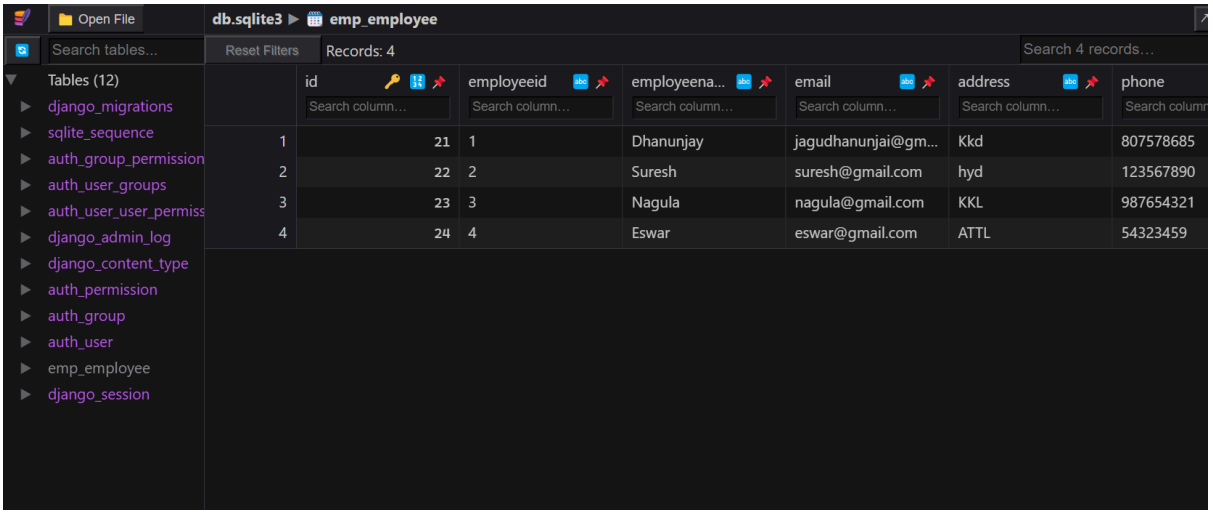
The delete functionality enables the removal of records from the database. Users can select a record to delete, and the application will confirm the deletion before removing the data.

Flow:

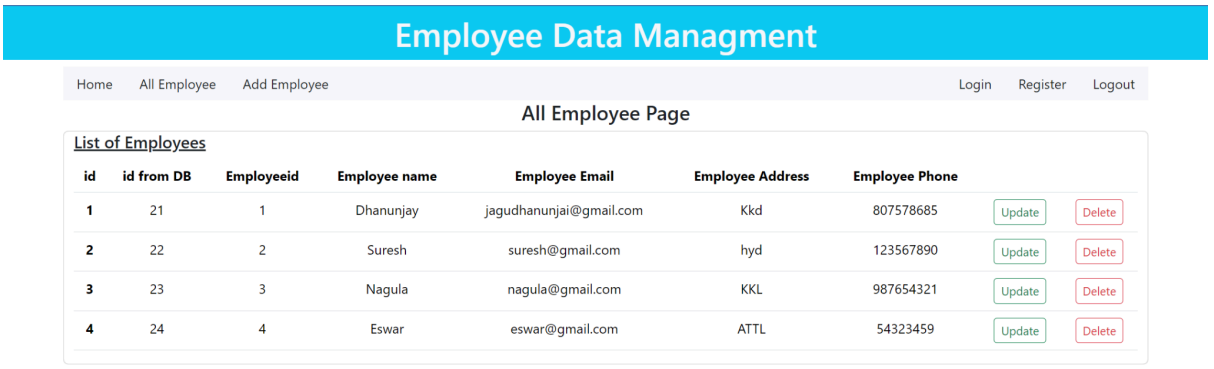
1. **Delete Action:** Deletes the selected record from the database.







2. **Redirection:** Redirects back to the home page upon successful deletion of the record.



Conclusion

This internship project involved the development of a Django CRUD application aimed at managing employee records. Through the implementation of Create, Read, Update, and Delete operations, the project successfully demonstrates the capabilities and features of the Django framework.

Key Accomplishments:

1. **Understanding Django Framework:**
    - Gained a comprehensive understanding of Django's Model-View-Template (MVT) architecture.
    - Successfully integrated Django ORM for seamless database interactions.
2. **Database Management:**
    - Designed and implemented a simple yet effective database schema using SQLite3.
    - Utilized Django ORM to perform CRUD operations, demonstrating efficient data management techniques.
3. **User Interface Development:**
    - Leveraged Bootstrap for responsive and user-friendly UI design.
    - Implemented various pages such as Home, Create, Read, Update, and Delete, providing a complete user experience for managing employee records.
4. **Practical Application:**
    - Developed a practical, real-world application that can be extended and enhanced for more complex use cases.
    - Laid the foundation for future enhancements, including user authentication, authorization, and advanced data relationships.

Future Enhancements:

While the current version of the application fulfills its primary objectives, there are several potential enhancements that can further improve its functionality and user experience:

- **User Authentication and Authorization:**
    - Implement user login and registration functionalities.
    - Add role-based access control to manage user permissions.
- **Advanced Data Relationships:**
    - Introduce more complex data models and relationships, such as department assignments and project allocations.
- **Enhanced UI/UX:**
    - Improve the overall user interface with more advanced Bootstrap components and custom styling.
    - Add client-side validation for form inputs to enhance user experience.

Learning Outcomes:

This project has been an invaluable learning experience, providing practical insights into web development with Django. Key learnings include:

- Mastering the MVT architecture and Django ORM for efficient database management.
- Understanding the importance of a clean, responsive design using HTML and Bootstrap.
- Gaining practical experience in debugging and optimizing a web application.

Conclusion:

The successful completion of this Django CRUD application marks a significant milestone in my internship project. The project not only meets its initial objectives but also provides a strong foundation for future development and enhancements. The skills and knowledge gained during this project will undoubtedly be beneficial in my future endeavors as a web developer.