Employee Management Application

Inhouse Management Software

2024

Ranjit Kumar Sahoo

Itio Inovex Pvt. Ltd.

3/19/2024

**About the Platform:**

The Employee Management system is a web-based application designed to streamline attendance tracking and management for employees within an organization. It offers features for both employees and administrators to efficiently manage attendance records, Update personal information and perform various administrative tasks.

**ADMIN PRIVILEGES:**

1. View Data:

* Admin has the responsibilities to view data related to Users, Employees, Department and Attendance records.
* Access to this information enables Admin to oversee and manage various aspects of the system.

1. Manage Data:

* Admin is responsible for creating, updating and deleting records for users, employees, departments and attendance data.
* This includes adding new users , updating employee details, creating new Department for Employee and managing attendance records effectively.

**EMPLOYEE PRIVILEGES:**

1. Attendance Management:

* Employees are responsible for marking their in-time and out-time accurately to maintain attendance records.
* Timely submission of attendance data ensures accurate tracking of work hours and adherence to organizational policies.

1. View Attendance Report:

* Employees have the capability to view their everyday attendance report including in-time, out-time and any absences.

1. Update Personal Detail:

* Employees can update their own details, including name, date of birth, profile pictures, address etc.
* This ensures that their personal information remains accurate and up-to-date in the system.

1. Change Password:

* Employees have the ability to change their password as needed for security purpose.
* Changing passwords provides employee with control over their account security and helps prevent unauthorized access.

1. Reset Forgot Password:

* In case of forgotten passwords, employees can initiate a password reset process.
* This capability allows employees to regain access to their accounts securely without Administrative intervention.

**TECHNOLOGY STACK:**

* **Backend Technology**: Django
* **Frontend Technology**: HTML, CSS, JavaScript, Bootstrap
* **Database**: SQLite
* **Security**: CSRF Protection, Session Management Authentication, Authorization, Hash Passwords.

**INSTALLATION:**

This installation guide provides step-by-step instructions for setting up and running a Django application locally. It covers environment setup, Software requirements, database setup(Using PostgreSQL), Project configuration and running the application on a local server.

**Environment Setup:**

1. Ensure you have a compatible operating system such as Windows, Linux or MacOS
2. Install Python from official website (<https://www.python.org/downloads/>) above than Python 3.

**Software Required:**

1. **Python**: Python Programming Language
2. **Django**: Web Framework for building web Application

**Project Configuration:**

1. **Create Virtual Environment:**

* Open a terminal or command prompt.
* Run the following command to create a virtual environment named virtual\_env

*python –m venv virtual\_env*

**Errors**:

* Permission issues or missing dependencies

**Error Handling**:

* Ensure Python is installed correctly.
* Write permission in the working directory.
* Verify the virtual environment module is available.

1. **Activate Virtual Environment:**

* Navigate to the virtual environment directory:

*cd virtual\_env*

* Activate the virtual environment:

On Windows

*Scripts\activate*

On MacOS and Linux

*source bin/activate*

**Errors:**

* Command not Found
* Activation script is not running

**Error Handling:**

* Recheck the path to the activation Script, Ensuring the compatibility with the user’s operating system.
* Troubleshooting any environment variable conflicts.

1. **Clone the Git Repository:**

* Run the following command to clone the Git repository:

*git clone* [*https://github.com/RanjitItio/Employee\_Management.git*](https://github.com/RanjitItio/Employee_Management.git)

**Errors:**

* Network connectivity issue
* Authentication Failure

**Error Handling:**

* Check network connectivity
* Verify repository permission, Ensure correct repository URL is used

1. **Install Required Packages:**

* Navigate to the cloned project directory:

*cd Employee\_Management*

* Install the required Python packages listed in the requirements.txt file:

*pip install -r requirements.txt*

**Errors:**

* Dependency Conflict
* Package installation Failure

**Error Handling:**

* Use virtual environment to isolate the dependencies.
* Updating ‘pip’ and ‘setuptools’ to the latest version

1. **Run the Local Development Server:**

* Start the local development server by running the following command:

*python manage.py runserver*

**Errors:**

* Port conflict
* Django Configuration Issue

**Error Handling:**

* Specify exact port number mentioned on the shell
* Review Django settings for any misconfigurations

1. **Access the Application:**

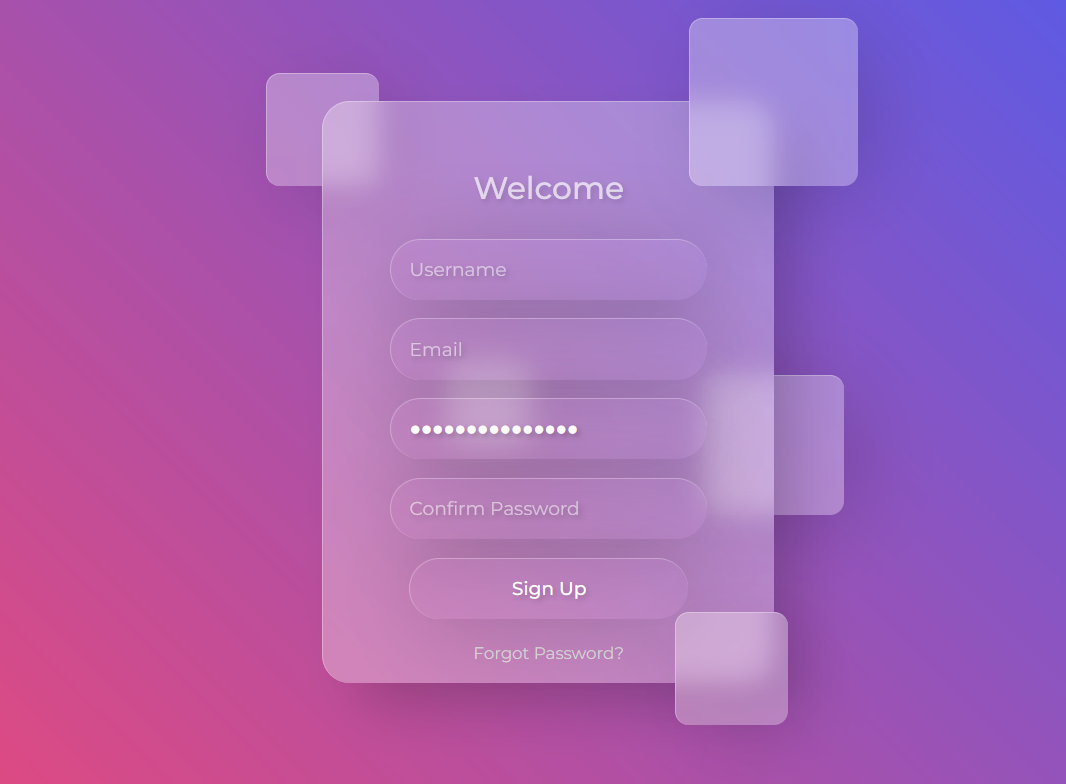
Open a web browser and enter the following URL:

[*http://127.0.0.1:8000/*](http://127.0.0.1:8000/)

USER JOURNEY:

USER REGISTRATION:

**Registration Page:**



**Coding Process:**

The RegisterView function is used to authenticate the user and login to the system.

**Parameters:**

request (HttpRequest): The incoming request object.

username (str): The username of the user.

email (str): The emailof the user.

password (str): The password of the user.

confirm\_password (str): The confirm\_password of the user.

**Returns:**

HttpResponse: An HTTP response object.

**Redirect**:  
 Redirect to the Login Page

**Raises:**

Success\_message (str): An success message to be displayed to the user after successful registration.

**ERRORS:**

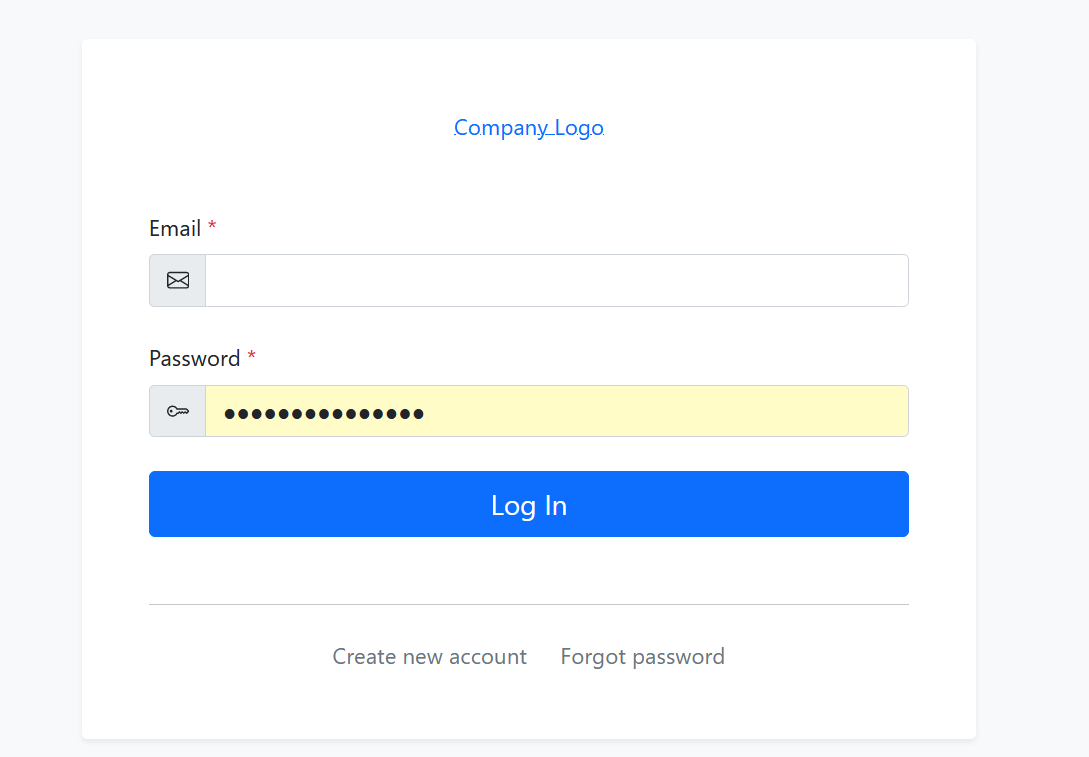
* Form submission without filling all the required fields
* Mismatch Password
* Error 403 Already Exists
* Form validation Error.
* Csrf verification failed

**ERROR HANDLING:**

* Fill in all the required fields
* Make sure to match both password fields
* Change the username and email if already exists or try to login
* Make sure to put a proper email format and password should contain Capital Letter, Small Letter, Special characters, Numeric value and minimum length of the password should be greater than 8.
* Reload the page or try in another browser.

USER LOGIN:

**Login Page:**



**Coding Process:**

The LoginView function is used to authenticate the user and login to the system.

**Parameters:**

request (HttpRequest): The incoming request object.

email (str): The email of the user.

password (str): The password of the user.

**Returns:**

HttpResponse: An HTTP response object.

**Raises:**

error\_message (str): An error message to be displayed to the user.

**ERRORS:**

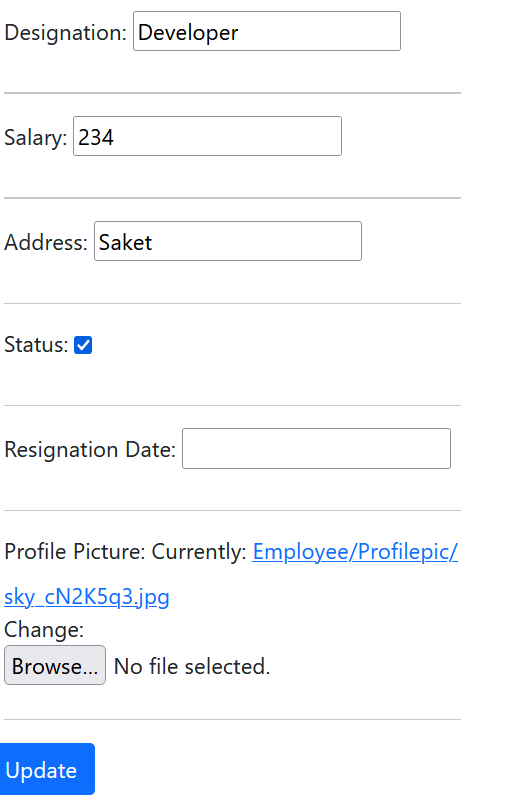
* Error 204 No Content
* Invalid email or Password
* Authentication Failure
* Session management error

**ERROR HANDLING:**

* Make sure to provide valid email and password
* Message will appear as Invalid email and password please recheck and try again
* Try again later or contact for support for issue
* Try to Login again to establish new connection

UPDATE USER PROFILE:

**Profile Update Page:**



**Coding Process:**

class EmployeeUpdateView(LoginRequiredMixin, UpdateView):

"""

View for updating employee details.

**Args**:

LoginRequiredMixin: A mixin that requires the user to be logged in.

UpdateView: A generic view for updating a model instance.

**Attributes**:

model: The model class for which the view updates instances.

template\_name: The name of the template to be used for rendering the view.

fields: The fields of the model to be included in the form.

success\_url: The URL to redirect to after a successful update.

**Methods**:

test\_func: Checks if the current user has permission to update the employee details.

**Returns**:

None

"""

model = EmployeeDetail

template\_name = 'Employee/empupdate.html'

fields = "\_\_all\_\_"

success\_url = '/emp/emp-home'

def test\_func(self):

"""

Checks if the current user has permission to update the employee details.

Returns:

True if the user is the employee or an admin, False otherwise.

"""

employee = self.get\_object()

if self.request.user == employee.employee:

return True

elif self.request.user.is\_staff:

return True

return False

**ERRORS:**

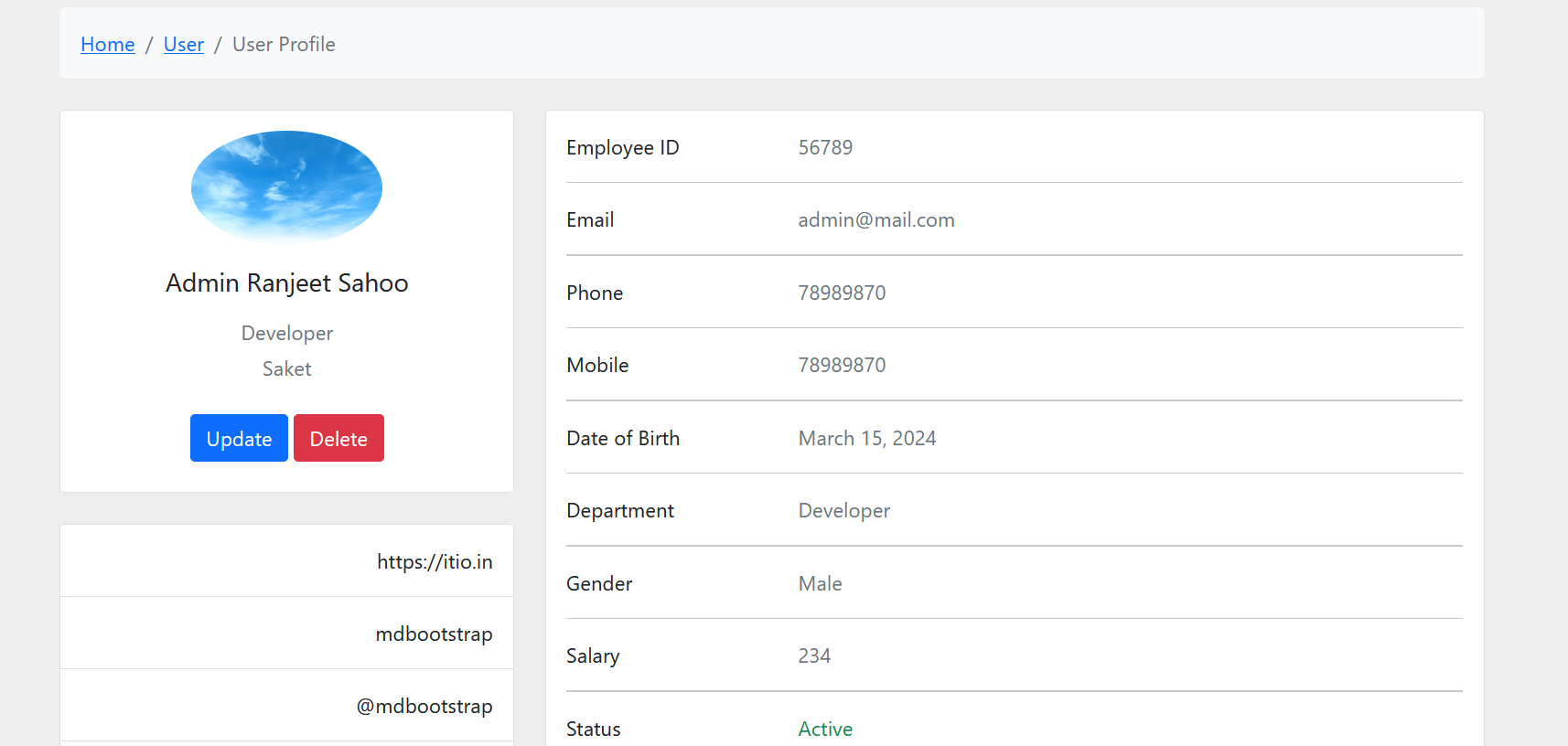
* Error 401 Unauthorized Access
* Error 404 Not found
* Error 403 Forbidden
* Form validation error

**ERROR HANDLING:**

* Make sure to Authenticate (Login) before accessing the page.
* Make sure to register as an Employee
* Accessed data must be of logged in user
* Provide proper data type in the fields

VIEW PROFILE DETAIL:

**Profile Detail Page:**



**Coding Process:**

class EmployeeDetailView(LoginRequiredMixin, DetailView):

"""

View for displaying employee details.

**Args**:

LoginRequiredMixin: A mixin that requires the user to be logged in.

DetailView: A generic view for displaying a model instance.

**Attributes**:

model: The model class for which the view displays instances.

template\_name: The name of the template to be used for rendering the view.

**Methods**:

test\_func: Checks if the current user has permission to view the employee details.

**Returns**:

None

"""

model = EmployeeDetail

template\_name = 'Employee/empdetail.html'

def test\_func(self, request, \*args, \*\*kwargs):

"""

Checks if the current user has permission to view the employee details.

Args:

request: The HTTP request object.

Returns:

True if the user is the employee, False otherwise.

"""

employee = self.get\_object()

if self.request.user == employee:

return True

return False

**ERRORS:**

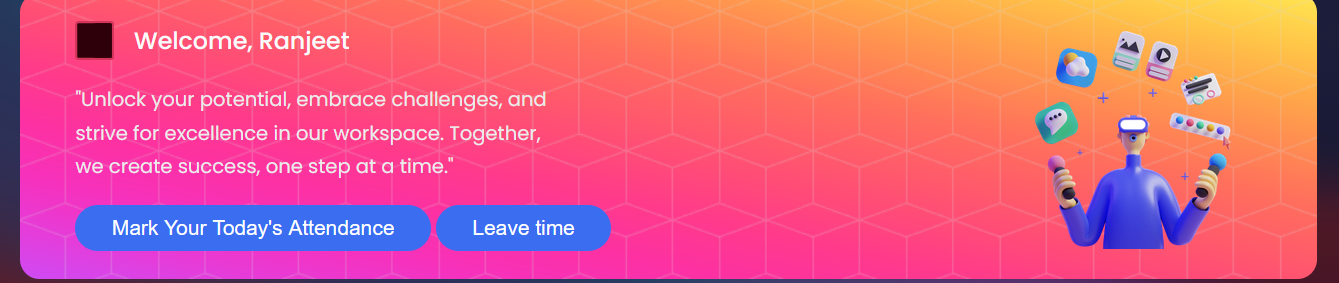
* Error 401 Unauthorized Access
* Error 404 Not Found
* Error 403 Permission Denied

**ERROR HANDLING:**

* Make sure to authenticate before accessing the page
* Must be registered as an Employee
* Accessing data must be of logged in user cannot view other Employee records

UPDATE IN-TIME ATTENDANCE:

**Attendance Page:**

****

**Coding Process:**

@login\_required

def IntimeAttendanceReportView(request):

"""

View for registering the in-time attendance of an employee.

**Args**:

request: The HTTP request object.

**Returns**:

If the request method is POST and the attendance is successfully registered, renders the attendance report page.

If the request method is GET, renders the in-time attendance form page.

"""

user = request.user

attendance\_report = None

employee\_detail = None

attendance\_report\_success = None

if request.method == 'POST':

# Attendance registration logic

Once the Attendance Marked by a user can not be remarked again.

...

else:

# Render the in-time attendance form page

...

context = {

'attendance\_report': attendance\_report,

'employee\_detail': employee\_detail,

'attendance\_report\_success': attendance\_report\_success

}

return render(request, 'Employee/intime.html', context)

**ERRORS:**

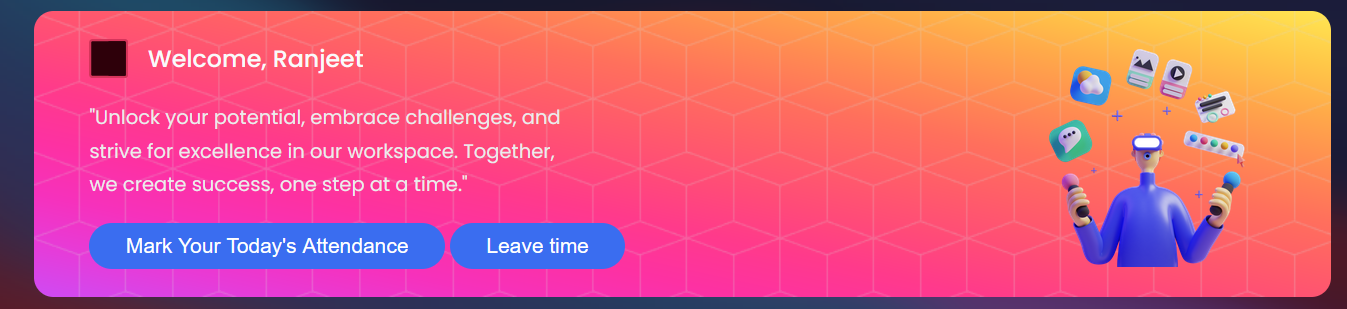
* Error 403 Forbidden
* Attendance already registered for the day
* Error 500 Internal server Error

**ERROR HANDLING:**

* Make sure to register as an Employee
* Once created cannot re register again for any query contact Admin department
* Please try again later

UPDATE OUT-TIME ATTENDANCE:

**Attendance Page:**



**Coding Process:**

def OutTimeAttendanceReportView(request, pk):

"""

View for registering the out-time attendance of an employee.

**Args**:

request: The HTTP request object.

pk: The primary key of the EmployeeAttendance object.

**Returns**:

If the request method is POST and the out-time attendance is successfully registered, renders the leave time report

Page.

If the request method is GET, renders the out-time attendance form page.

"""

user = request.user

leavetime\_report = None

leavetime\_report\_success = None

if request.method == "POST":

# Out-time attendance registration logic

Once marked can not be remarked again until next Day

...

else:

# Render the out-time attendance form page

...

context = {

'leavetime\_report': leavetime\_report,

'leavetime\_report\_success': leavetime\_report\_success

}

return render(request, 'Employee/outime.html', context)

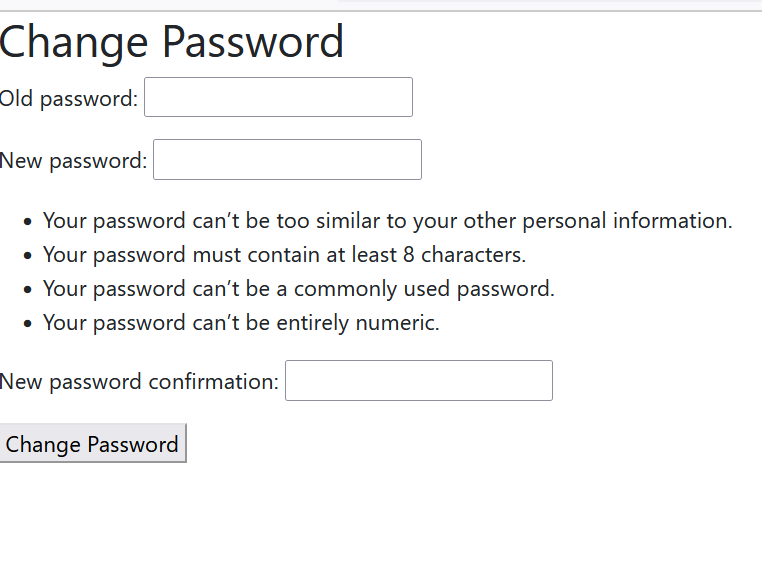
**ERRORS:**

* Error 400 Bad request
* Error 403 Permission Denied
* Error 500 Internal server Error

**ERROR HANDLING:**

* Provide correct primary key of a Employee Attendance record
* Cannot re assign Attendance once registered
* Please try again later

CHANGE PASSWORD:

**Change password page:**  
**Coding Process:**

@login\_required

def ChangePasswordView(request):

"""

View for changing the password of the logged-in user.

**Args**:

request: The HTTP request object.

**Returns**:

If the request method is POST and the password change is successful, redirects to the login page.

If the request method is GET, renders the change password form page.

"""

if request.method == **'POST'**:

# Password change logic

Check previous password is correct or not if the previous password is incorrect

return response “password\_incorrect”

if the previous password is correct it will change the password

...

else:

# Render the change password form page

...

context = {

'form': form

}

return render(request, 'Users/change\_password.html', context)

**ERROR:**

* Invalid Credential
* Session update failure
* Invalid data type

**ERROR HANDLING:**

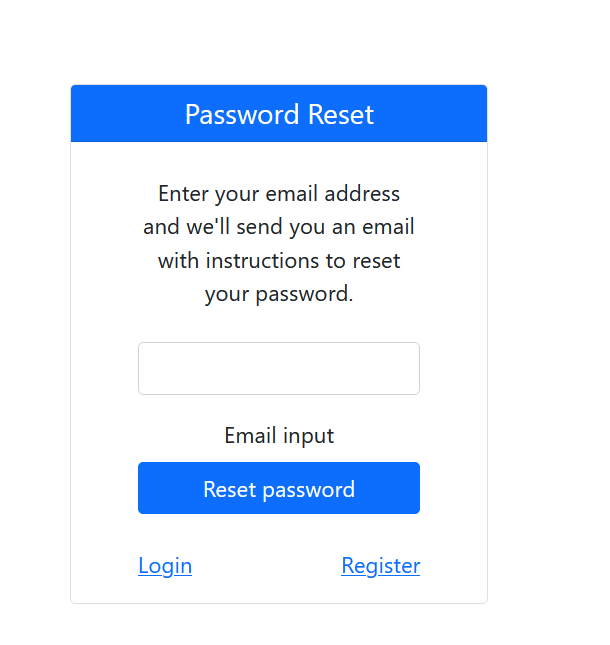
* Make sure the old password is correct and password and confirm password field should match
* Reload the page and try again
* Password format must be according to the given instruction

RESETTING FORGOTTEN PASSWORD:

**STEPS INVOLVED:**

1. Fill the Email Address
2. Email Notification
3. Set New Password
4. Success Message
5. **FILL THE EMAIL ADDRESS:**

**Email Address Filling Page:**



**Coding Process:**

@login\_required

def PasswordResetView(request):

"""

View for initiating the password reset process.

**Args**:

request: The HTTP request object.

**Returns**:

If the request method is POST and the email is registered, sends a password reset email and redirects to the password reset confirmation page.

If the request method is GET, renders the password reset form page.

"""

if request.method == "POST":

# Password reset logic

Create a uid and Password Reset token for using the user ID and send in mail to identify the user and the token will

remain valid for 15 minute for security purpose.

...

else:

# Render the password reset form page

...

return render(request, 'Users/password\_reset.html')

**ERRORS:**

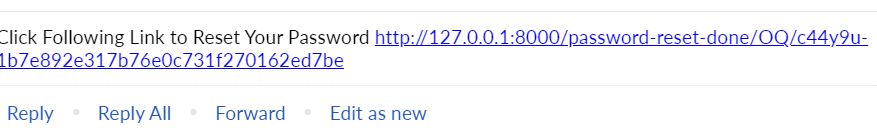
* Error 204 No Content
* Error 404 Not Found
* Token Generation Failure
* Mail sent failure

**ERROR HANDLING:**

* Provide proper email in the input field
* Register the email address before proceeding to next step
* Retry again
* Try after some time

1. **EMAIL NOTIFICATION:**

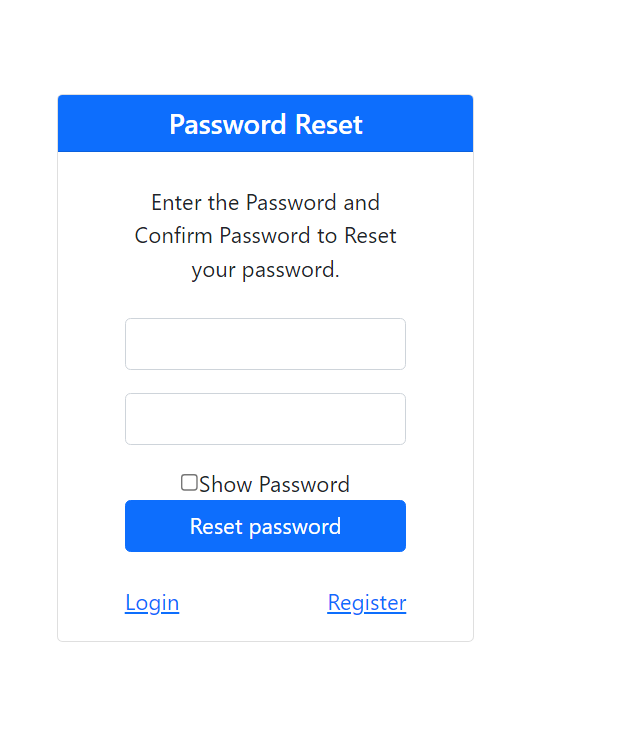
**Email Notification Message:**



An email will be sent to the provided email address containing a unique link to reset the password.

1. **SET NEW PASSWORD:**

**Set New Password Page:**



**Coding Process:**

def PasswordResetDoneView(request, uid, token):

"""

View for completing the password reset process.

**Args**:

request: The HTTP request object.

uid: The user ID encoded in base64.

token: The password reset token.

**Returns**:

If the request method is POST and the password reset is successful, renders the password reset confirmation page.

If the request method is GET, renders the password reset completion form page.

"""

if request.method == "POST":

# Password reset completion logic

If the password and confirm\_password matched and Password Reset Token has not Expired then set new

Password for the user.

...

else:

# Render the password reset completion form page

...

return render(request, 'Users/password\_reset\_done.html')

**ERRORS:**

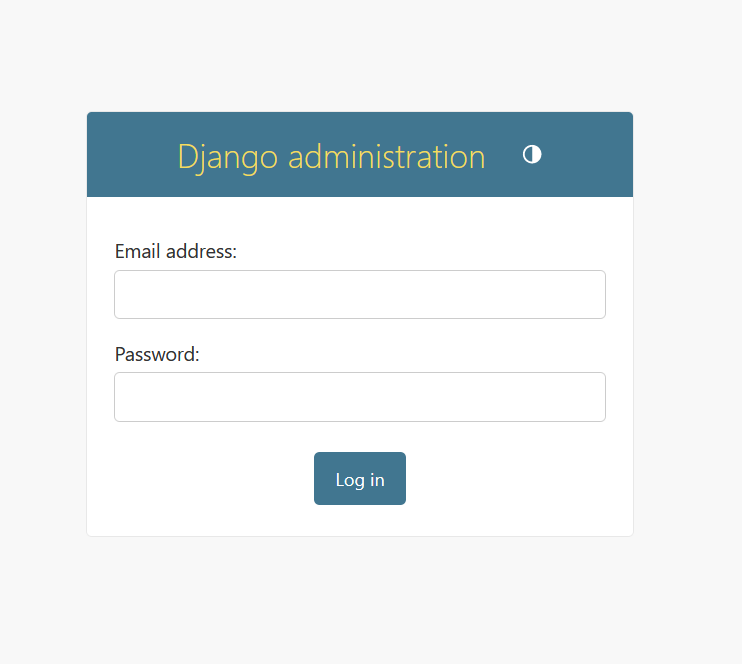
* Password Mismatch
* Invalid Token or Expired Token
* Invalid User ID
* Internal Server error

**ERROR HANDLING:**

* Make sure to provide same password In both fields
* Reset the password within 15 minutes of mail sent else the token will expire
* Uid must be of a user and generated by server
* Please try again after some time

ADMIN JOURNEY:

**Admin Login Page:**



**Coding Process:**

**Description**:

The login method is responsible for rendering the login form for the admin interface. It first checks if the user is already logged in and redirects to the admin index if so. Otherwise, it renders the login form using Django's built-in authentication views.

The admin login process in Django is facilitated by the **django.contrib.admin.sites.AdminSite.login** method. This method allows administrators to securely authenticate themselves and access the administrative interface for managing Django models.

**Parameters**:

self: The instance of the admin site.

request: The HTTP request object.

extra\_context: Additional context data to be passed to the login template.

**Returns**:

An HTTP response containing the rendered login form.

**Dependencies**:

never\_cache method decorator.

AdminAuthenticationForm from django.contrib.admin.forms.

LoginView from django.contrib.auth.views.

**Usage Notes**:

This method is typically used within a custom admin class for a Django model.

**Security Considerations:**

Ensure that the never\_cache decorator is applied to prevent caching of sensitive information

@admin.register(CustomUser)

class CustomUserAdmin(admin.ModelAdmin):

def login(self, request, extra\_context=None):

"""

Display the login form for the given HttpRequest.

"""

if request.method == "GET" and self.has\_permission(request):

# Already logged-in, redirect to admin index

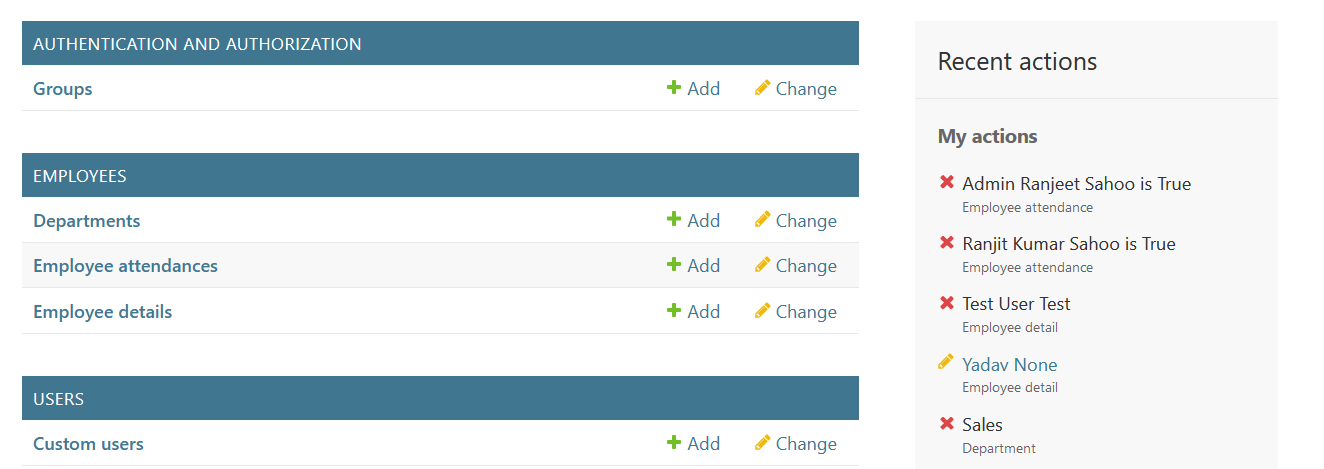
index\_path = reverse("admin:index", current\_app=self.name)

return HttpResponseRedirect(index\_path)

return LoginView.as\_view(\*\*defaults)(request)

**ADMIN HOME PAGE:**

Admin Home Page Image:



**Coding Process:**

@user\_passes\_test(lambda u: u.is\_staff, login\_url='/login/')

@login\_required

def HomePage(request):

"""

View for rendering the home page.

***Args****:*

request: The HTTP request object*.*

***Returns****:*

If the user is authenticated, renders the home page with employee details and attendance information.

If the user is not authenticated, renders the home page without any employee details or attendance information.

"""

user = request.user

employee\_id = None

employee\_detail = None

employee\_attendance\_id = None

if user.is\_authenticated:

# Retrieve employee details and attendance information

...

else:

employee\_id = None

employee\_detail = None

context = {

'employee\_id': employee\_id,

'employee\_detail': employee\_detail,

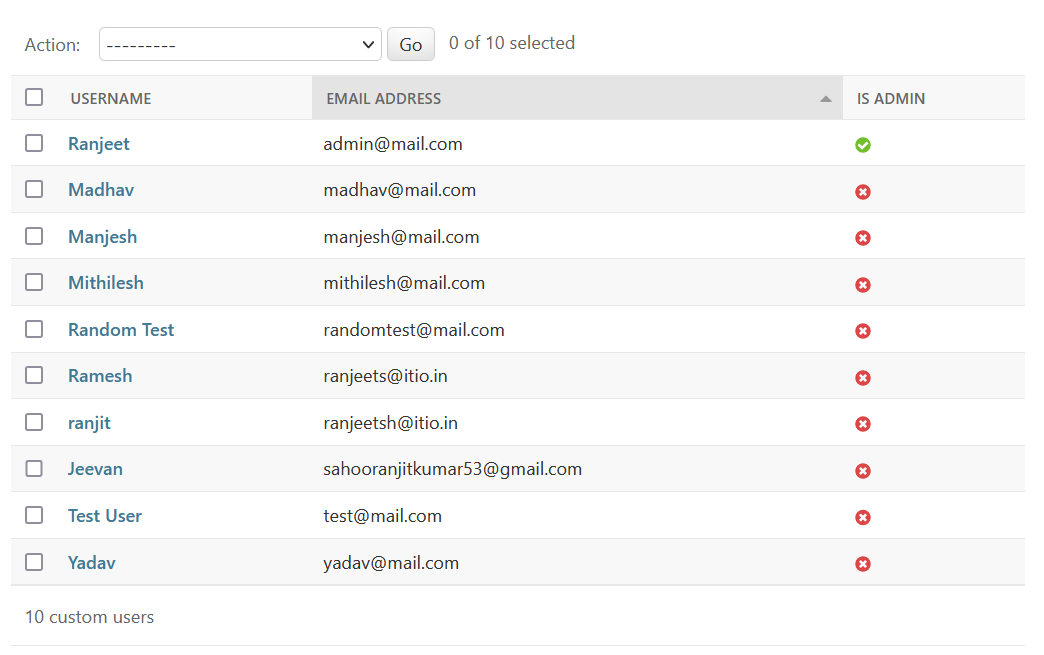
'employee\_attendance\_id': employee\_attendance\_id

}

return render(request, 'Users/home.html', context)

***ALL USER INTERFACE:***

***All User Interface Page:***



**Coding Process:**

@csrf\_protect\_m

def changelist\_view(self, request, extra\_context=None):

"""

The 'change list' admin view for this model.

**Args**:

request: The HTTP request object.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the change list view.

"""

from django.contrib.admin.views.main import ERROR\_FLAG

app\_label = self.opts.app\_label

if not self.has\_view\_or\_change\_permission(request):

raise PermissionDenied

try:

cl = self.get\_changelist\_instance(request)

except IncorrectLookupParameters:

# Wacky lookup parameters were given, so redirect to the main

# changelist page, without parameters, and pass an 'invalid=1'

# parameter via the query string. If wacky parameters were given

# and the 'invalid=1' parameter was already in the query string,

# something is screwed up with the database, so display an error

# page.

if ERROR\_FLAG in request.GET:

return SimpleTemplateResponse(

"admin/invalid\_setup.html",

{

"title": \_("Database error"),

},

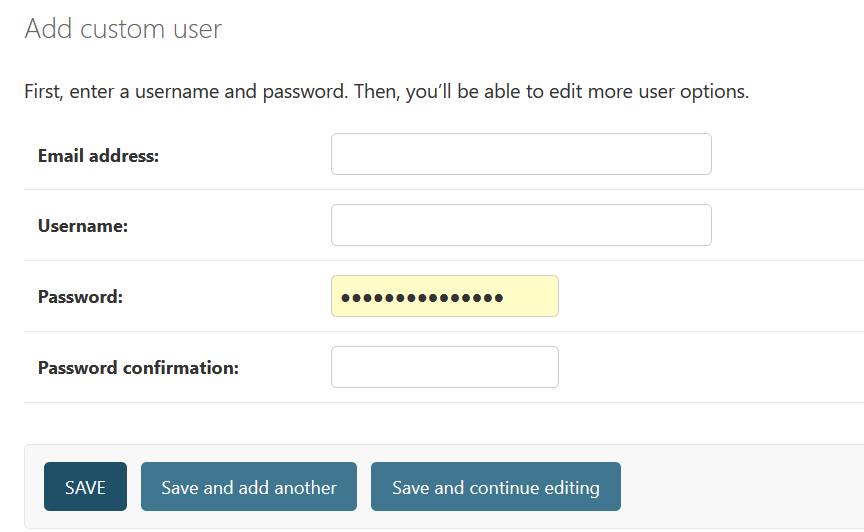
)

return HttpResponseRedirect(request.path + "?" + ERROR\_FLAG + "=1")

# Rest of the code...

**CREATE NEW USER INTERFACE:**

**Create New User Page:**



**Coding Process:**

@sensitive\_post\_parameters\_m

@csrf\_protect\_m

def add\_view(self, request, form\_url="", extra\_context=None):

"""

View for displaying the add form for a new model instance.

**Args**:

request: The HTTP request object.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the add view.

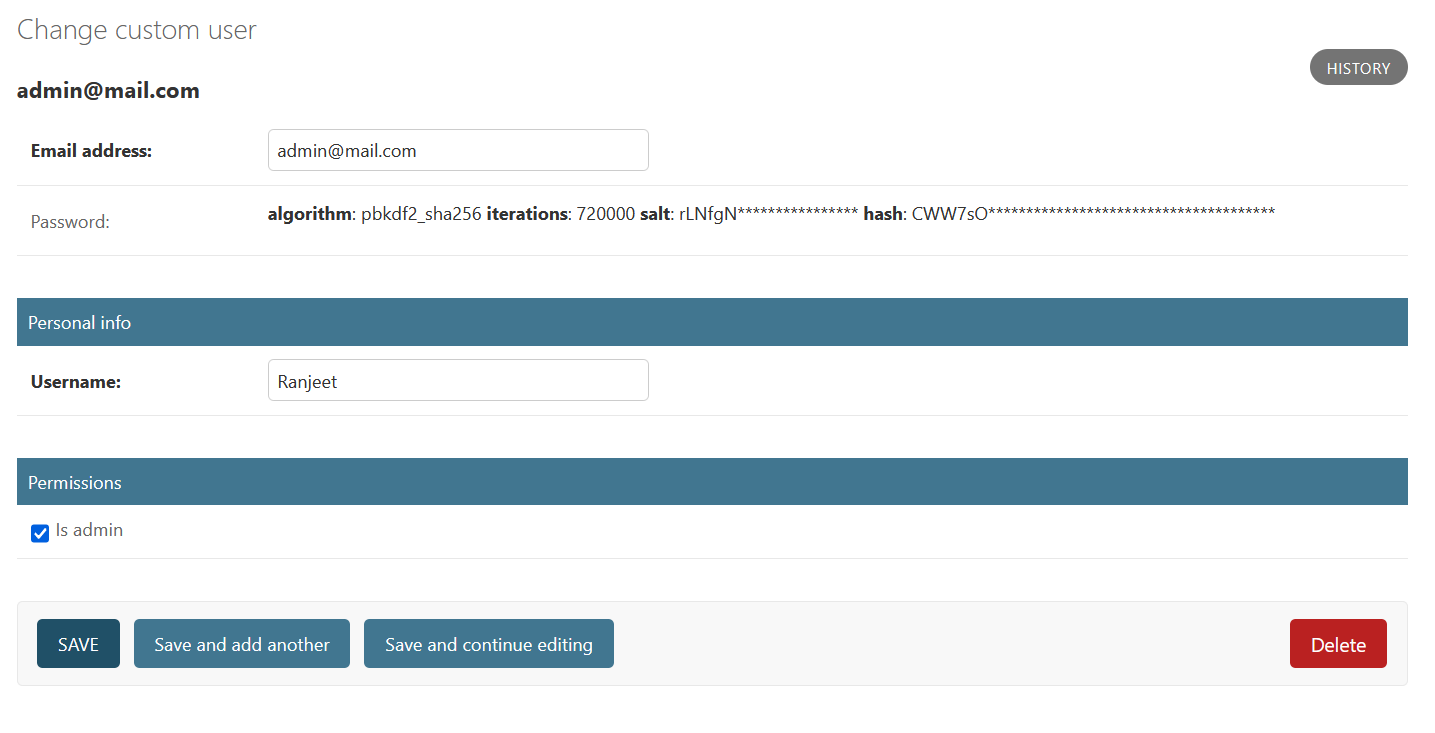
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_add\_view(request, form\_url, extra\_context)

**UPATE A USER DETAIL:**

**Update user page:**

****

**Coding Process:**

def change\_view(self, request, object\_id, form\_url="", extra\_context=None):

"""

View for displaying the change form for an existing model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be changed.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

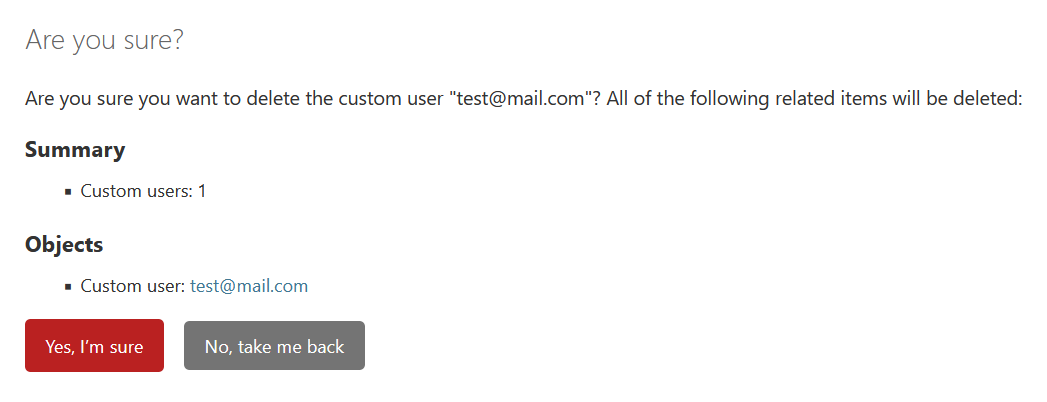
The rendered template response for the change view.

"""

return self.changeform\_view(request, object\_id, form\_url, extra\_context)

**DELETE USER INTERFACE:**

**Delete user page:**

****

**Coding Process:**

@csrf\_protect\_m

def delete\_view(self, request, object\_id, extra\_context=None):

"""

View for displaying the delete confirmation page for a model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be deleted.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the delete view.

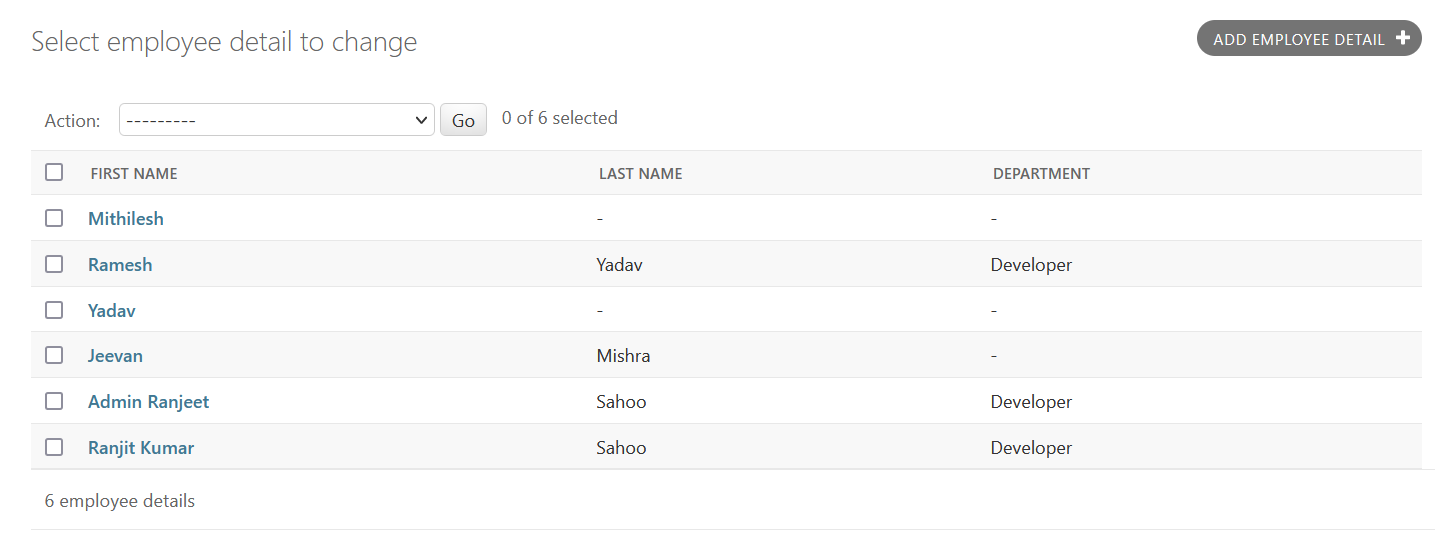
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_delete\_view(request, object\_id, extra\_context)

**ALL EMPLOYEE INTERFACE:**

**All Employee Page:**



**Coding Process:**

def changelist\_view(self, request, extra\_context=None):

"""

The 'change list' admin view for this model.

**Args**:

request: The HTTP request object.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the change list view.

"""

from django.contrib.admin.views.main import ERROR\_FLAG

app\_label = self.opts.app\_label

if not self.has\_view\_or\_change\_permission(request):

raise PermissionDenied

try:

cl = self.get\_changelist\_instance(request)

except IncorrectLookupParameters:

# Wacky lookup parameters were given, so redirect to the main

# changelist page, without parameters, and pass an 'invalid=1'

# parameter via the query string. If wacky parameters were given

# and the 'invalid=1' parameter was already in the query string,

# something is screwed up with the database, so display an error

# page.

if ERROR\_FLAG in request.GET:

return SimpleTemplateResponse(

"admin/invalid\_setup.html",

{

"title": \_("Database error"),

},

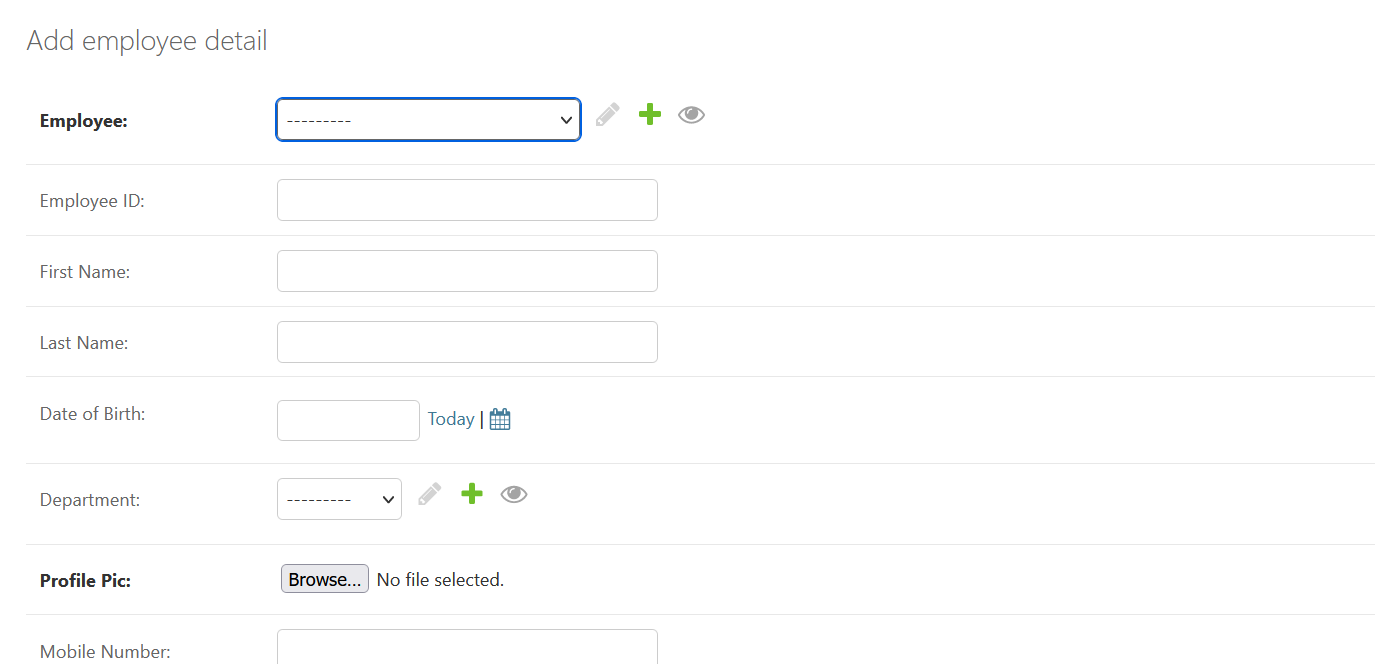
)

return HttpResponseRedirect(request.path + "?" + ERROR\_FLAG + "=1")

# Rest of the code...

**CREATE NEW EMPLOYEE:**

**Create new employee page:**



**Coding Process:**

@sensitive\_post\_parameters\_m

@csrf\_protect\_m

def add\_view(self, request, form\_url="", extra\_context=None):

"""

View for displaying the add form for a new model instance.

**Args**:

request: The HTTP request object.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the add view.

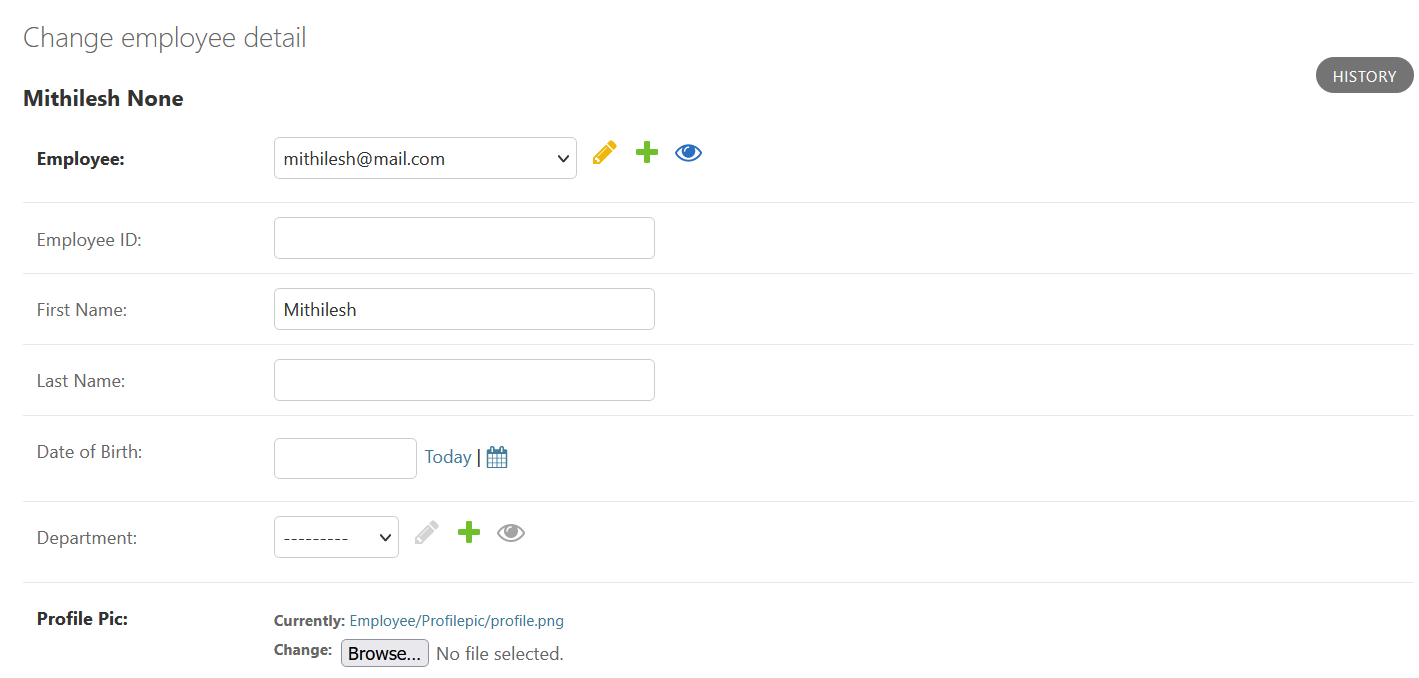
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_add\_view(request, form\_url, extra\_context)

**UPDATE NEW EMPLOYEE:**

**Update new employee detail page:**



**Coding Process:**

def change\_view(self, request, object\_id, form\_url="", extra\_context=None):

"""

View for displaying the change form for an existing model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be changed.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

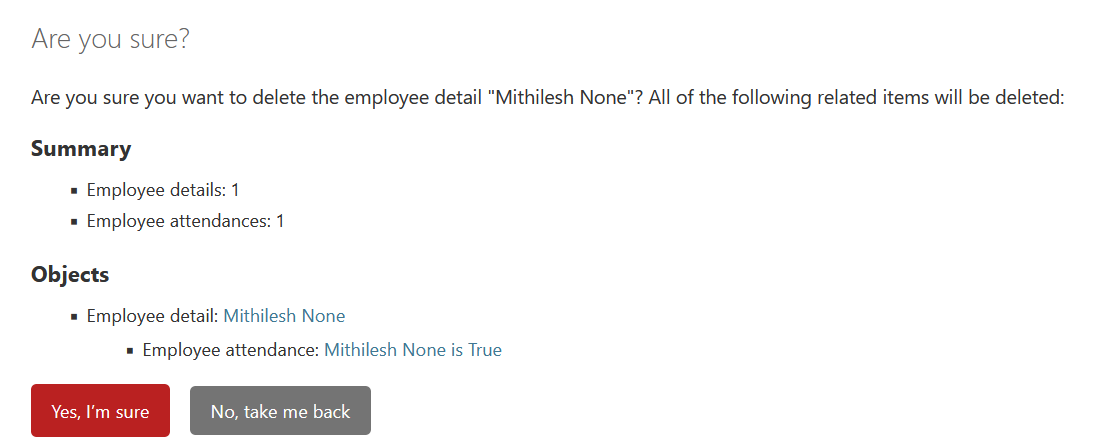
The rendered template response for the change view.

"""

return self.changeform\_view(request, object\_id, form\_url, extra\_context)

**DELETE AN EMPLOYEE:**

**Delete an employee page:**



**Coding Process:**

@csrf\_protect\_m

def delete\_view(self, request, object\_id, extra\_context=None):

"""

View for displaying the delete confirmation page for a model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be deleted.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the delete view.

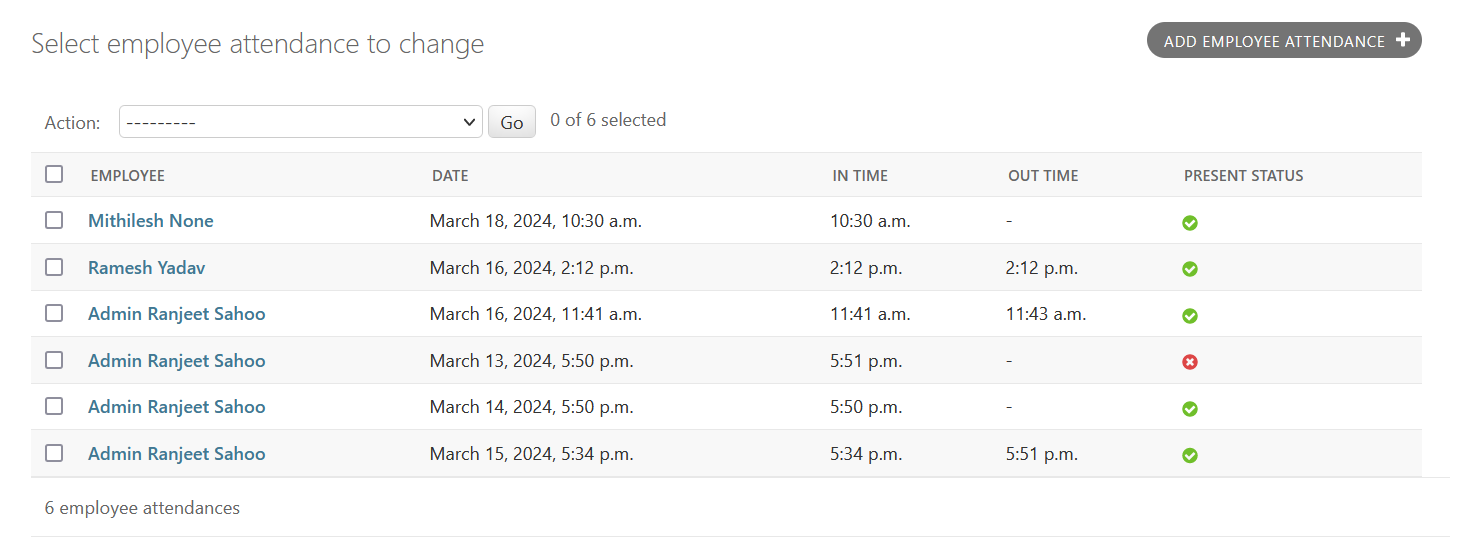
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_delete\_view(request, object\_id, extra\_context)

**EMPLOYEE ATTENDANCE INTERFACE:**

**Employee attendance page:**



**Coding Process:**

@csrf\_protect\_m

def changelist\_view(self, request, extra\_context=None):

"""

The 'change list' admin view for this model.

**Args**:

request: The HTTP request object.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the change list view.

"""

from django.contrib.admin.views.main import ERROR\_FLAG

app\_label = self.opts.app\_label

if not self.has\_view\_or\_change\_permission(request):

raise PermissionDenied

try:

cl = self.get\_changelist\_instance(request)

except IncorrectLookupParameters:

# Wacky lookup parameters were given, so redirect to the main

# changelist page, without parameters, and pass an 'invalid=1'

# parameter via the query string. If wacky parameters were given

# and the 'invalid=1' parameter was already in the query string,

# something is screwed up with the database, so display an error

# page.

if ERROR\_FLAG in request.GET:

return SimpleTemplateResponse(

"admin/invalid\_setup.html",

{

"title": \_("Database error"),

},

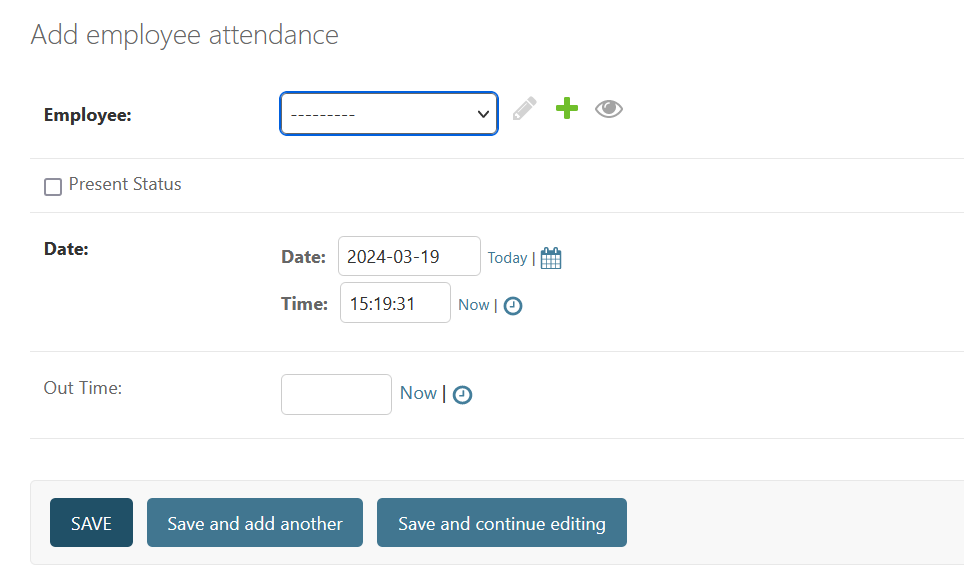
)

return HttpResponseRedirect(request.path + "?" + ERROR\_FLAG + "=1")

# Rest of the code...

**CREATE NEW EMPLOYEE ATTENDANCE:**

**Create attendance page:**



**Coding Process:**

@sensitive\_post\_parameters\_m

@csrf\_protect\_m

def add\_view(self, request, form\_url="", extra\_context=None):

"""

View for displaying the add form for a new model instance.

**Args**:

request: The HTTP request object.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the add view.

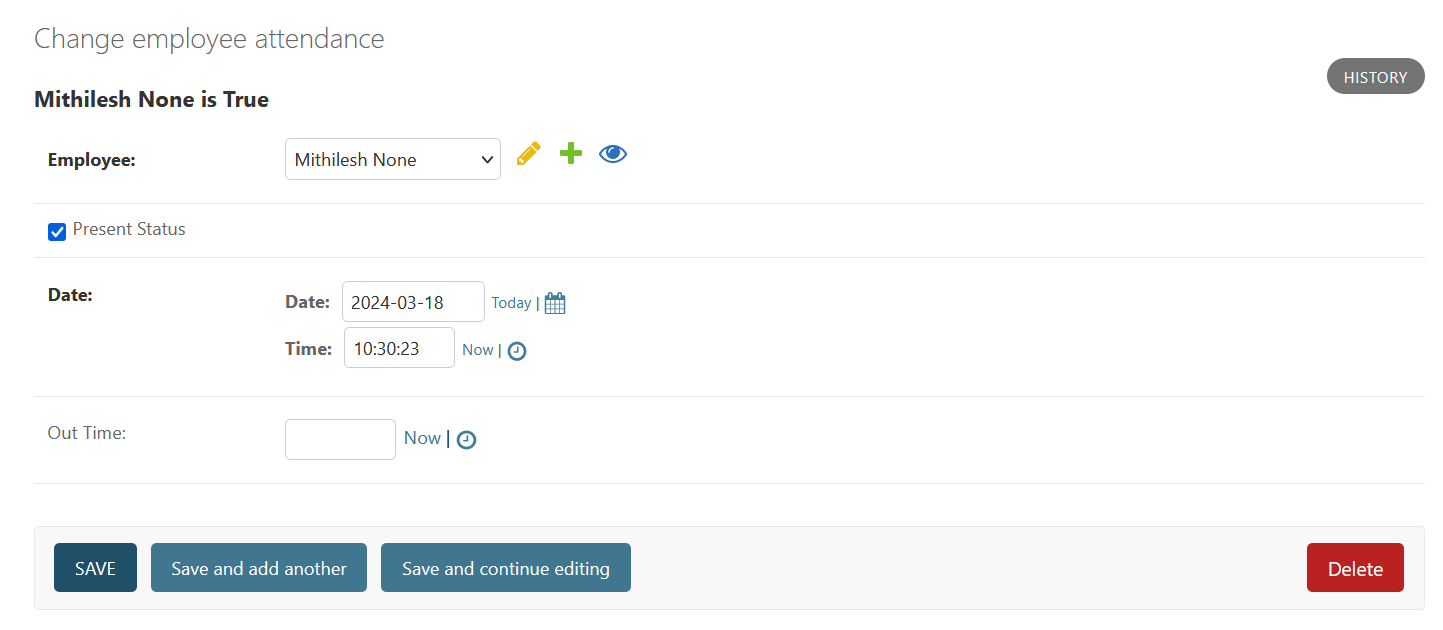
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_add\_view(request, form\_url, extra\_context)

**UPDATE EMPLOYEE ATTENDANCE REPORT:**

**Update attendance report page:**



**Coding Process:**

def change\_view(self, request, object\_id, form\_url="", extra\_context=None):

"""

View for displaying the change form for an existing model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be changed.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

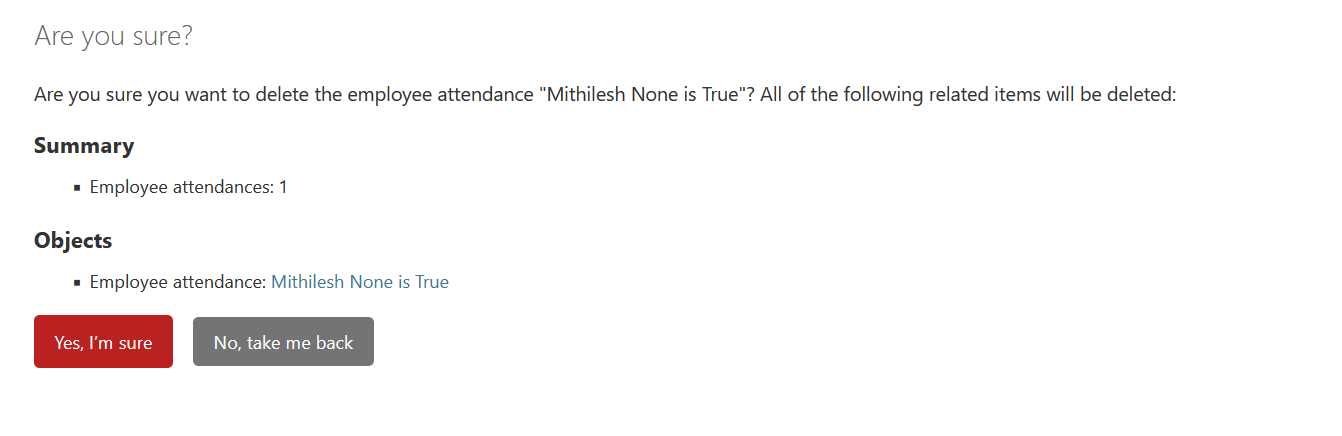
The rendered template response for the change view.

"""

return self.changeform\_view(request, object\_id, form\_url, extra\_context)

**DELETE EMPLOYEE ATTENDANCE REPORT:**

**Delete attendance report page:**



**Coding Process:**

@csrf\_protect\_m

def delete\_view(self, request, object\_id, extra\_context=None):

"""

View for displaying the delete confirmation page for a model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be deleted.

extra\_context: Additional context data (optional).

**Returns**:

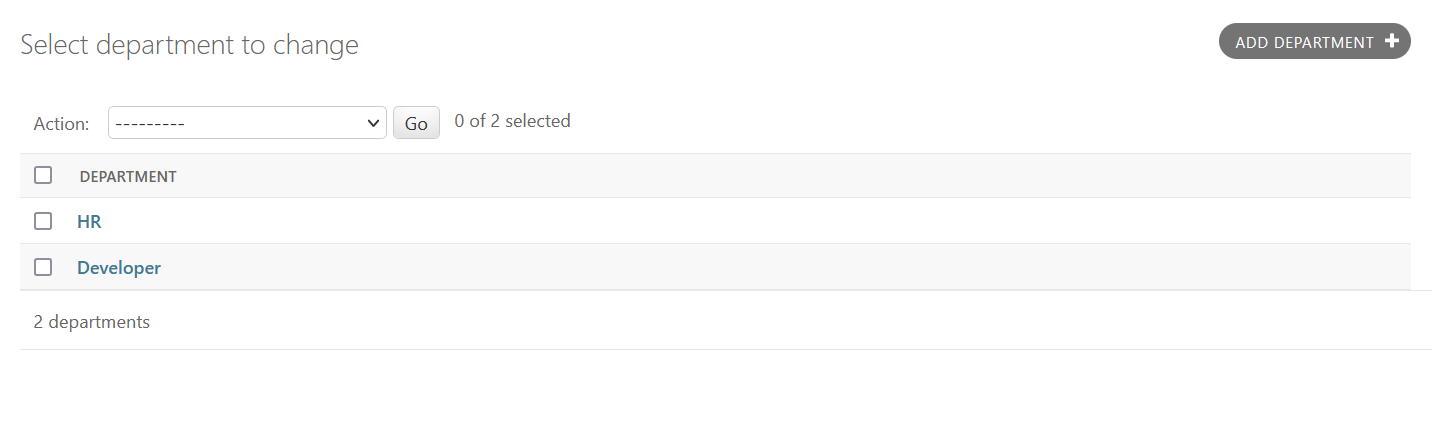
The rendered template response for the delete view.

"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_delete\_view(request, object\_id, extra\_context)

**ALL DEPARTMENT SECTION**  
**All department page:**



**Coding Process:**

@csrf\_protect\_m

def changelist\_view(self, request, extra\_context=None):

"""

The 'change list' admin view for this model.

**Args**:

request: The HTTP request object.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the change list view.

"""

from django.contrib.admin.views.main import ERROR\_FLAG

app\_label = self.opts.app\_label

if not self.has\_view\_or\_change\_permission(request):

raise PermissionDenied

try:

cl = self.get\_changelist\_instance(request)

except IncorrectLookupParameters:

# Wacky lookup parameters were given, so redirect to the main

# changelist page, without parameters, and pass an 'invalid=1'

# parameter via the query string. If wacky parameters were given

# and the 'invalid=1' parameter was already in the query string,

# something is screwed up with the database, so display an error

# page.

if ERROR\_FLAG in request.GET:

return SimpleTemplateResponse(

"admin/invalid\_setup.html",

{

"title": \_("Database error"),

},

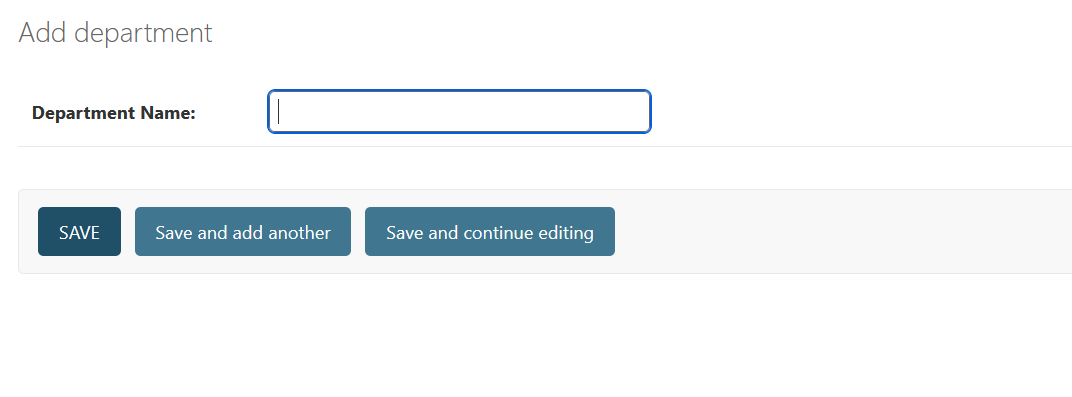
)

return HttpResponseRedirect(request.path + "?" + ERROR\_FLAG + "=1")

# Rest of the code...

**CREATE NEW DEPARTMENT:**

**Create new department page:**



**Coding process:**

@sensitive\_post\_parameters\_m

@csrf\_protect\_m

def add\_view(self, request, form\_url="", extra\_context=None):

"""

View for displaying the add form for a new model instance.

**Args**:

request: The HTTP request object.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the add view.

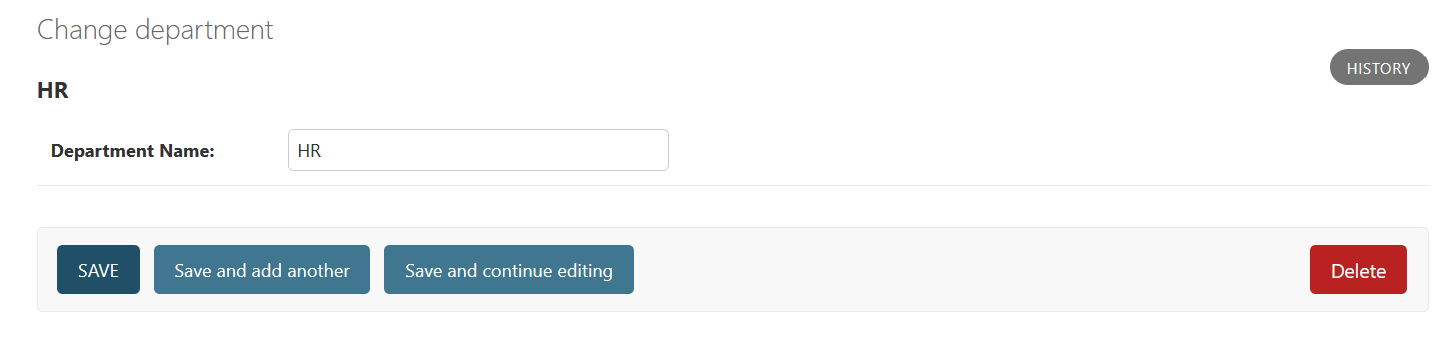
"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_add\_view(request, form\_url, extra\_context)

**UPDATE AN EXISTING DEPARTMENT:**

**Update department page:**



**Coding Process:**

def change\_view(self, request, object\_id, form\_url="", extra\_context=None):

"""

View for displaying the change form for an existing model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be changed.

form\_url: The URL to submit the form to (optional).

extra\_context: Additional context data (optional).

**Returns**:

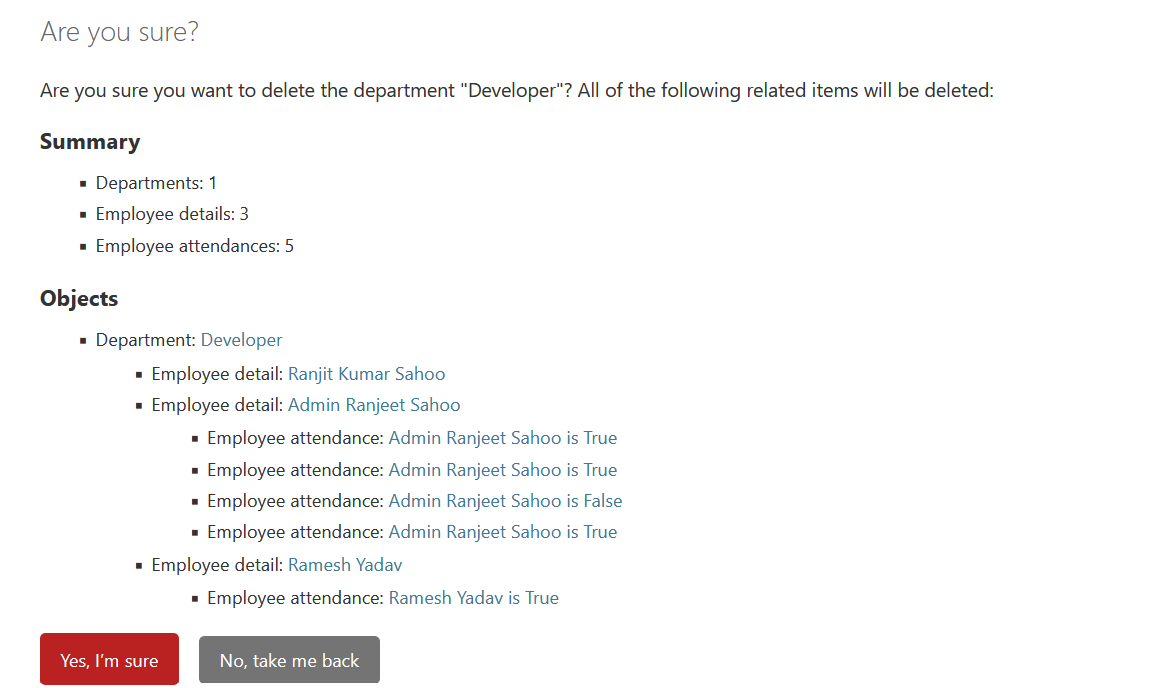
The rendered template response for the change view.

"""

return self.changeform\_view(request, object\_id, form\_url, extra\_context)

**DELETE ANY DEPARTMENT:**

**Delete department page:**



**Coding Process:**

@csrf\_protect\_m

def delete\_view(self, request, object\_id, extra\_context=None):

"""

View for displaying the delete confirmation page for a model instance.

**Args**:

request: The HTTP request object.

object\_id: The ID of the model instance to be deleted.

extra\_context: Additional context data (optional).

**Returns**:

The rendered template response for the delete view.

"""

with transaction.atomic(using=router.db\_for\_write(self.model)):

return self.\_delete\_view(request, object\_id, extra\_context)