**Detailed Design**

**for**

**Task Management System**

**Capstone:**

**Prepared by**

**COP-2939-93453**

**Instructor: Dr. Banisakher**

**Date: October 9, 2024**

***Preliminary Design Page ii***

Contents

[1. Introduction 1](#_Toc179977629)

[2. System Architecture Description 3](#_Toc179977630)

[3. Detailed Design of Major Components 5](#_Toc179977631)

[4. API Design 7](#_Toc179977632)

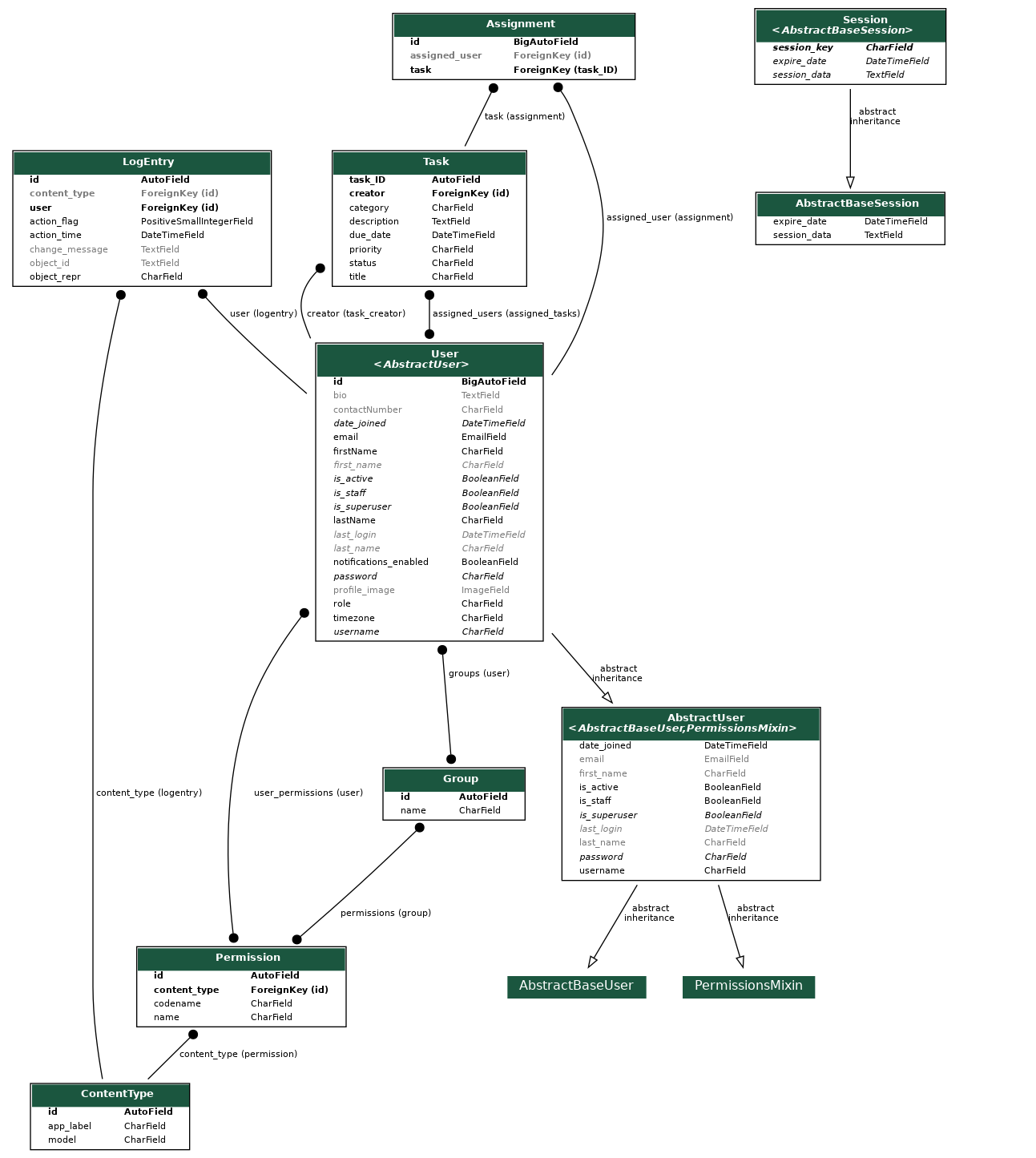
[5. User Interface UI design 8](#_Toc179977633)

[6. Security Considerations 11](#_Toc179977634)

[7. Error Handling 11](#_Toc179977635)

[8. Testing Strategy 12](#_Toc179977636)

# Introduction

The **Task Management System** is a web-based application designed to facilitate the creation, management, and assignment of tasks across different user roles (Admin, Manager, and User). This system enables the tracking of task statuses, user assignments, and deadlines. The platform also supports user authentication, notifications, and role-based access controls.

# System Architecture Description

2.1 High-Level Architecture

The system follows a **client-server architecture**, with Django serving as the backend framework and a responsive HTML/CSS frontend with potential for JavaScript enhancement. The system will integrate with PostgreSQL/MySQL as the primary database for data storage.

* **Client-Side (Frontend):** HTML/CSS/JavaScript for rendering dynamic pages and forms.
* **Server-Side (Backend):** Django handles user authentication, task creation, assignment management, and other business logic.
* **Database:** PostgreSQL stores users, tasks, assignments, and status data.

The application will follow a **Model-View-Controller (MVC)** pattern where:

* **Model**: Represents the database schema using Django’s ORM.
* **View**: Manages the display of information to users via templates.
* **Controller**: Manages user requests, processes them, and returns appropriate responses.

2.2 Database Design

**Entities:**

* **User:** Stores user information such as name, role, email, password, and bio. Relationships include tasks they created or are assigned to.
* **Task:** Captures the title, description, due date, status, priority, category, and assignment information.
* **Assignment:** Connects tasks to users, tracking who is responsible for each task.

**ER Diagram:**

A diagram of a company

Description automatically generated

2.3 Data Flow

1. **User logs in:** The user authentication is handled by Django's built-in authentication system.
2. **Task Creation:** Users create tasks, and a task is linked to the creator
3. **Task Assignment:** Admin/Manager can assign tasks to users.
4. **Task Tracking:** Users can track tasks by status (Incomplete, In Progress, Completed).
5. **Notifications:** Users are notified when new tasks are assigned or when deadlines are near.

2.4 Component Descriptions

1. **Django Views**: Controllers responsible for processing user requests and returning appropriate responses.
2. **Django Models**: Representations of database entities (e.g., User, Task, Assignment).
3. **Forms**: Used for user input validation when creating or updating tasks.
4. **Templates**: HTML files rendered with dynamic content using Django's templating engine.

# Detailed Design of Major Components

3.1 User Management

Users are categorized into different roles:

1. **Admin**: Full system access to create and manage tasks and users.
2. **Manager**: Can assign tasks to users and view progress reports.
3. **User**: Can view and update assigned tasks.

User data is stored in a custom Django model extending AbstractUser:

class User(AbstractUser):

role = models.CharField(max\_length=20, choices=[('admin', 'Admin'), ('manager', 'Manager'), ('user', 'User')])

profile\_image = models.ImageField(upload\_to='profile\_images/', null=True, blank=True)

contactNumber = models.CharField(max\_length=15, null=True, blank=True)

3.2 Task Management

Tasks are represented using a Django model:

class Task(models.Model):

title = models.CharField(max\_length=250)

description = models.TextField()

due\_date = models.DateTimeField()

priority = models.CharField(max\_length=10, choices=[('High', 'High'), ('Medium', 'Medium'), ('Low', 'Low')])

status = models.CharField(max\_length=20, choices=[('Incomplete', 'Incomplete'), ('In Progress', 'In Progress'), ('Completed', 'Completed')])

creator = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='created\_tasks')

3.3 Assignment Management

The assignment of tasks to users is managed via the Assignment model”

class Assignment(models.Model):

task = models.ForeignKey(Task, on\_delete=models.CASCADE)

assigned\_user = models.ForeignKey(User, on\_delete=models.CASCADE)

3.4 Dashboard Reports

The dashboard will show:

1. **Assigned tasks**: Tasks assigned to the logged-in user.
2. **Created tasks**: Tasks created by the user.
3. **Task status breakdown**: A summary of tasks grouped by status.

A screenshot of a computer

Description automatically generated

# API Design

A RESTful API can be implemented for external systems or mobile apps. The API will expose endpoints like:

* **GET /tasks/**: Retrieve all tasks.
* **POST /tasks/**: Create a new task.
* **PUT /tasks/{id}**: Update a task.
* **DELETE /tasks/{id}**: Delete a task.

# User Interface UI design

The user interface will be designed with a clean and intuitive layout. Key pages include:

* **Login Page**: Allows users to authenticate.

A screenshot of a login form

Description automatically generated

* **Dashboard**: Displays tasks and their statuses.

A screenshot of a computer

Description automatically generated

* **Task Creation**: A form to create new tasks.

A screenshot of a computer

Description automatically generated

* **Task Update**: A form to update existing tasks.

A screenshot of a computer

Description automatically generated

# Security Considerations

Django uses a multilayered security policy, these policies are as shown:

1. **Authentication**: Django's authentication framework will handle user login and session management.
2. **Authorization**: Role-based access control will be used to restrict access to certain functionalities.
3. **Data Protection**: Sensitive user data (e.g., passwords) will be hashed using Django's default password hashing mechanism.

# Error Handling

Error handling will be managed using Django's built-in error views and custom error handling middleware. Key error types include:

* **404 Not Found**: When a task or user is not found.
* **403 Forbidden**: Unauthorized access attempts.
* **500 Server Error**: General server-side errors.

# Testing Strategy

Testing will be carried out using Django’s TestCase class, with both unit and integration tests. Key areas to be tested:

* **User authentication**.
* **Task creation, update, and deletion**.
* **Assignment logic**.
* **Dashboard and reporting**.