**Preliminary Design**

**for**

**Task Management System**

**Capstone:**

**Prepared by**

**COP-2939-93453**

**Instructor: Dr. Banisakher**

**Date: October 9, 2024**

***Preliminary Design Page ii***

Table of Contents

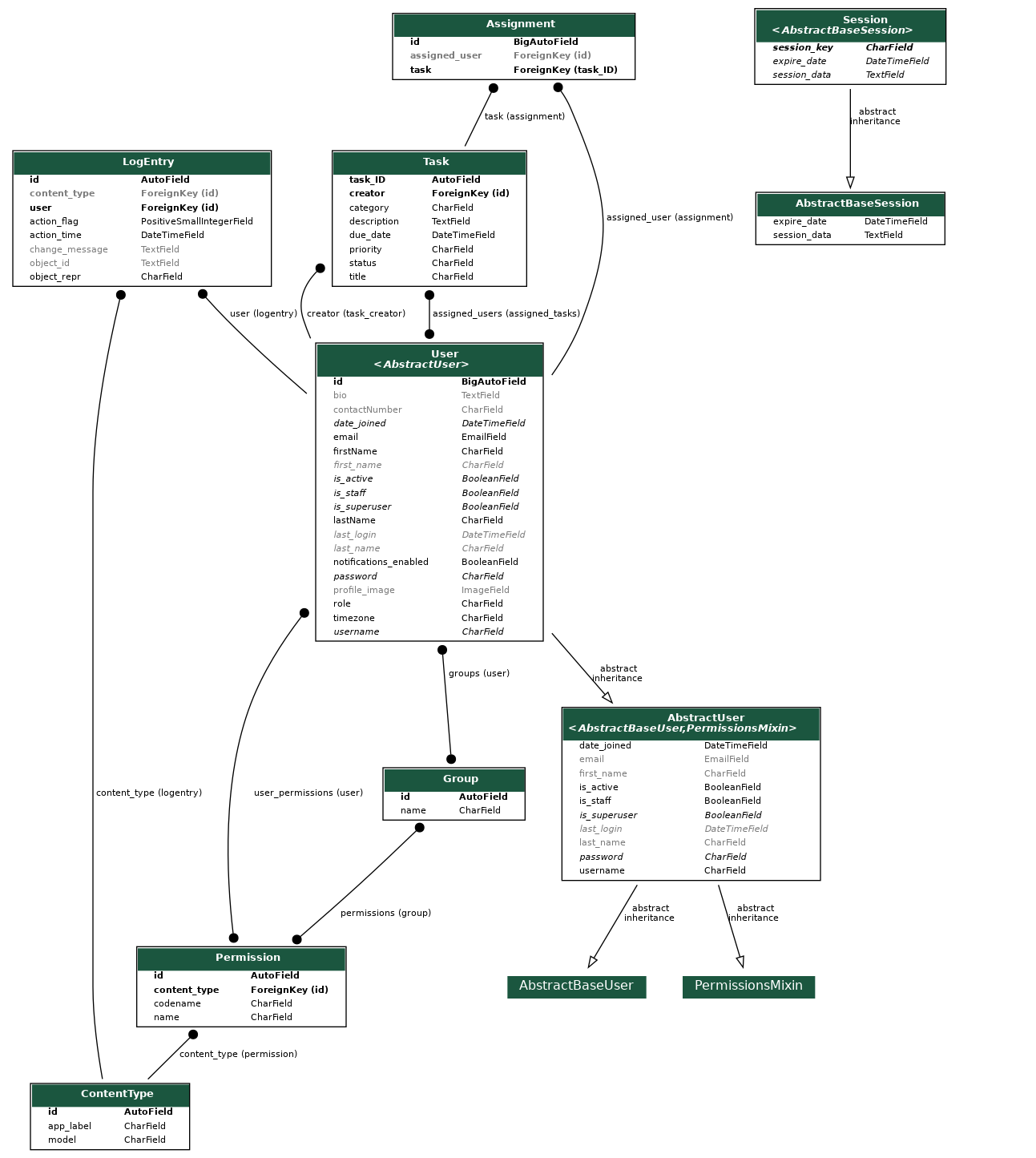
[1. Overview 1](#_Toc179358331)

[2. System Architecture Description 3](#_Toc179358332)

3. Frameworks and Software …..……………………………………………………………………………………………………………4

# Overview

The purpose of this document is to provide a preliminary design for the Task Management System, which will serve as the capstone project for COP-2939. This system is intended to assist users in managing tasks effectively by providing features like task creation, assignment, tracking, and user management. The design described here will provide a foundation for the final system implementation and testing.



# System Architecture Description

2.1 Overall Structure

The system architecture will follow a **three-tier architecture** consisting of:

1. **Presentation Layer**: This is the frontend that the user interacts with. It will be built using HTML, CSS, and JavaScript, rendered by Django templates. The navigation and interaction with tasks, users, and role-based features will occur here.
2. **Application Logic Layer**: This layer handles the business logic of the system, implemented with Django as the web framework. It will manage user authentication, task creation and management, user roles, and permission handling.
3. **Data Layer**: The data layer consists of a PostgreSQL database where user, task, and role data are stored. This layer manages the relationships between users and tasks, as well as the integrity of the 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 Breakdown of Each Routine

**User Management**:

* **Routine**: create\_user()
  + Description: Allows new users to register, providing credentials, role selection, and profile image upload.
  + Input: username, password, role
  + Output: New user entry in the database.
* **Routine**: assign\_task()
  + Description: Managers or Admins can assign tasks to users.
  + Input: task\_ID, assigned\_users
  + Output: Updates the assigned\_users field of the task.

**Task Management**:

* **Routine**: create\_task()
  + Description: Users can create tasks with a title, description, and due date.
  + Input: title, description, due\_date, priority
  + Output: Task entry in the database.
* **Routine**: view\_tasks()
  + Description: Allows users to view tasks they have created or tasks assigned to them.
  + Input: user\_id
  + Output: List of tasks filtered by user role and ownership.

**Role-Based Access Control (RBAC)**:

* **Routine**: check\_permission()
  + Description: Ensures users can only perform actions allowed by their roles.
  + Input: user\_role
  + Output: Boolean value, indicating access permission.

# Frameworks and Software

The development of the Task Management System will utilize the following frameworks and software:

**Backend Framework**:

* **Django** (Python-based framework) will serve as the backbone of the system. Django provides a robust ORM for database management and built-in user authentication mechanisms.

**Frontend**:

* **HTML5/CSS3/JavaScript** for the UI and **Django Templates** to integrate dynamic content.
* **Bootstrap** will be used for responsive design to ensure the application is mobile-friendly.

**Database**:

* **PostgreSQL** will store user information, task data, and role assignments. PostgreSQL is chosen for its reliability and integration with Django.

**Version Control**:

* **GitHub** will be used for version control, allowing collaborative development and version tracking.

**Deployment**:

* The system will be deployed using **Heroku** or **AWS**, making it accessible through the web.

**Security**:

* **Django’s built-in security features** will be used to protect against common vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
* **SSL** will be enforced for secure communication between clients and the server.