# Task Tracking Management System

NAME 1

NAME 2

NAME 3

# Problem Statement

| | |
|---|---|
| **Inefficient Task Tracking:** | Many teams and individuals currently rely on manual or disjointed methods for tracking tasks, leading to inefficiencies and a lack of clear oversight. |
| **Lack of Centralized System:** | Without a centralized platform, it's challenging to assign tasks, set deadlines, and monitor progress, resulting in decreased productivity and potential project delays. |
| **Difficulty in Collaboration:** | Collaboration on tasks often requires multiple communication channels. The absence of a unified system complicates the sharing of updates and feedback. |
| **Inadequate Task Prioritization:** | Prioritizing tasks based on urgency and importance is crucial. However, traditional systems don’t offer an effective way to manage task priorities. |
| **Limited Accessibility and Visibility:** | Teams need real-time access to task statuses and updates. Limited visibility into task progression hinders effective decision-making. |

# Proposed Solution

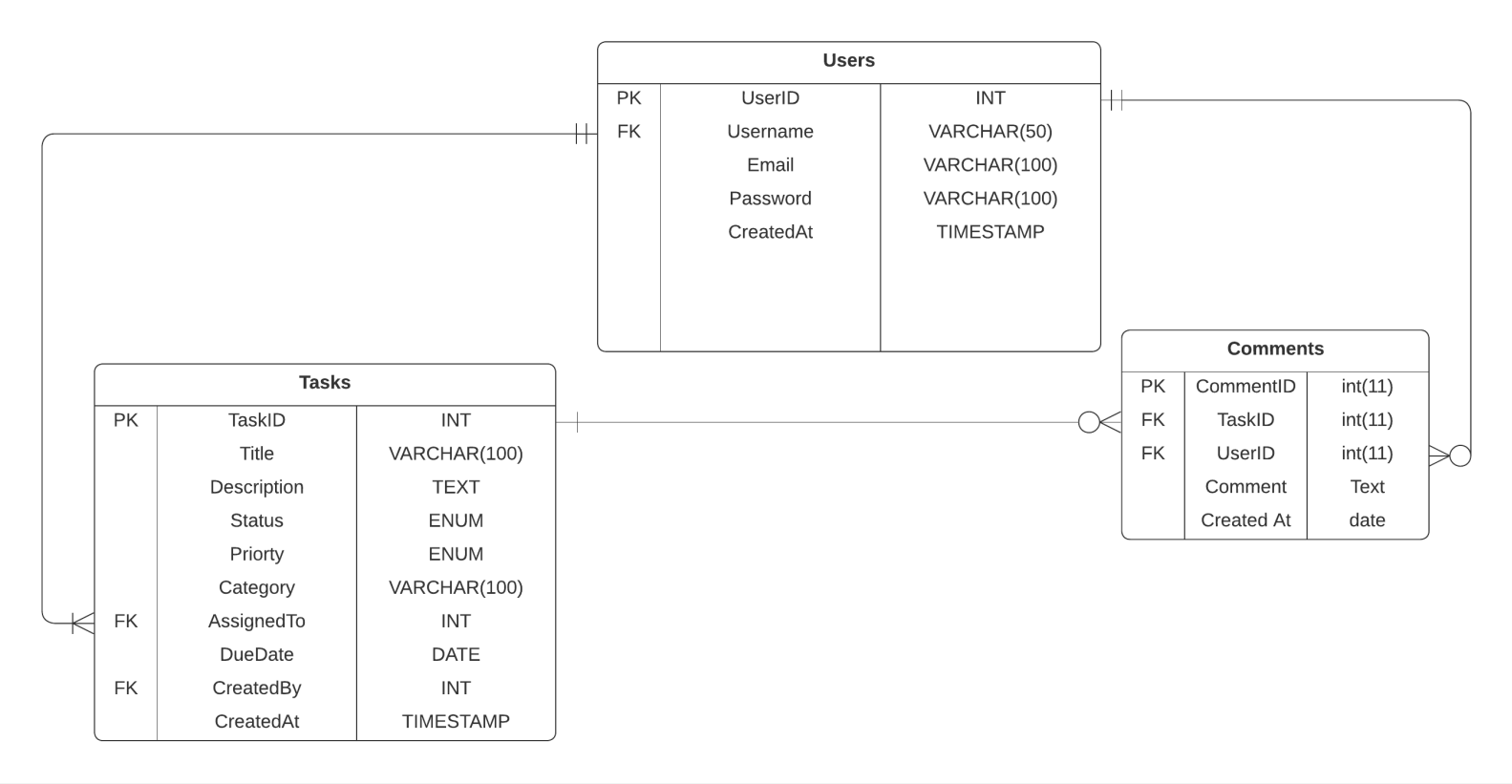
- Our project aims to address these challenges by developing an all-encompassing task management system that:
  1. Streamlines task tracking and management.
  2. Provides a centralized platform for task assignment, updates, and collaboration.
  3. Enhances team productivity and efficiency through intuitive design and user-friendly features.
  4. Facilitates effective task prioritization and project planning.
  5. Offers real-time visibility into task status and progress for all team members.

**Goal**: To create a more organized, efficient, and collaborative work environment for teams and individuals.

# Requirements

- **Hardware Requirements:**
  - OS: Windows(10 or more), Mac OS, Linux
  - Memory: 1 GB RAM or higher for smooth operation.
  - Processor: Intel Core i5 or equivalent for efficient processing.
- **Software Requirements:**
  - Frontend: Dynamic and user-friendly interface, built with the latest HTML5, CSS3, and JavaScript technologies.
  - Backend: Employs powerful Python web framework called Flask, which provides robust server-side logic. Additional Python dependencies ensure that the app is secure and reliable.
  - Database: AMPP with MySQL, which is a popular and well-respected choice for web applications.

# ER DIAGRAM


Users
PK UserID INT
FK Username VARCHAR(50)
Email VARCHAR(100)
Password VARCHAR(100)
CreatedAt TIMESTAMP
Tasks
PK TaskID INT
Title VARCHAR(100)
Description TEXT
Status ENUM
Priorty ENUM
Category VARCHAR(100)
FK AssignedTo INT
DueDate DATE
FK CreatedBy INT
CreatedAt TIMESTAMP
Comments
PK CommentID int(11)
FK TaskID int(11)
FK UserID int(11)
Comment Text
Created At date

# Entities & Relationships:

1. **Users:**
    1. Primary Entity representing system users.
    2. Fields:
        1. UserID: Unique identifier for each user (Primary Key).
        2. Username: The name chosen by the user.
        3. Email: User's email address.
        4. Password: Encrypted password for user authentication.
        5. CreatedAt: Timestamp of user creation.

# Entities & Relationships:

**2. Tasks:**

1. Core Entity representing the tasks to be managed.
2. Fields:
    1. TaskID: Unique identifier for each task (Primary Key).
    2. Title: Brief title of the task.
    3. Description: Detailed description of the task.
    4. Status: Current status of the task (e.g., Pending, In Progress, Completed).
    5. Priority: Priority level of the task (e.g., Low, Medium, High).
    6. Category: Categorization of the task.
    7. AssignedTo: UserID of the user to whom the task is assigned (Foreign Key).
    8. CreatedBy: UserID of the user who created the task (Foreign Key).
    9. DueDate: The date by which the task should be completed.
    10. CreatedAt: Timestamp of task creation.

# Entities & Relationships:

### 3. Comments:

1. Supporting Entity for additional information on tasks.
2. Fields:
    1. CommentID: Unique identifier for each comment (Primary Key).
    2. TaskID: Identifier of the task to which the comment belongs (Foreign Key).
    3. UserID: Identifier of the user who made the comment (Foreign Key).
    4. Comment: The text content of the comment.
    5. Created At: Date the comment was made.

# Relationships:

- **One-to-Many** between Users and Tasks: A single user can create and be assigned to many tasks.
- **One-to-Many** between Users and Comments: A user can make many comments, but each comment is made by one user.
- **One-to-Many** between Tasks and Comments: A task can have many comments, but each comment is associated with one task.

**Integrity Constraints:**

- Foreign keys ensure referential integrity between tasks and users, as well as comments and their associated tasks and users.

# Users Table

# Tasks Table

# Comments Table

# Front End

- **User Interface (UI) Design**:
  - Clean and intuitive interface for seamless user interaction.
  - Responsive design ensuring compatibility across devices and screen sizes.
- **Framework Utilization**:
  - Bootstrap for consistent and modern UI components.
  - Jinja2 templating for dynamic content rendering.
- **User Experience (UX)**:
  - Interactive elements like modals for task creation and edits.
  - Real-time form validations to guide user inputs.
  - Streamlined navigation for task management, including viewing, creating, and updating tasks.
  - Task cards on dashboard for quick access to task details.
  - Kanban-style dashboard for a visual snapshot of task progression.

# Front End – Login/ Signup Pages

- Put two images of login and signup pages

# Front End – Home Page

# Front End – Tasks Dashboard

# Front End – Users

# Front End – Create User

# Front End – Create Task

# Front End – Edit Task & Add/Delete Comments

# Backend – Bird's Eye View

- Robust SQL database models for Users, Tasks, and Comments.
- Relationships and integrity constraints to maintain data consistency.
- RESTful routes for CRUD operations on tasks and users.
- Secure endpoints for user authentication and authorization.
- Server-side processing for task assignments, updates, and priority management.
- Comment system for enhancing collaboration among users.
- Hashed passwords and token-based user sessions.
- Employ Flask SQL-Alchemy for database interactions and model representation.
- Utilize Flask-Login for handling user sessions with efficiency and reliability.
- Password Encryption: Implement Flask-Bcrypt for robust password encryption, safeguarding user credentials.

# Backend – Database Schema and SQLAlchemy Models

```sql
CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Username VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    Password VARCHAR(100) NOT NULL,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Tasks (
    TaskID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(100) NOT NULL,
    Description TEXT,
    Status ENUM('Pending', 'In Progress', 'Completed') NOT NULL DEFAULT 'Pending',
    Priority ENUM('Low', 'Medium', 'High') NOT NULL DEFAULT 'Medium',
    Category VARCHAR(100),
    AssignedTo INT,
    DueDate DATE,
    CreatedBy INT,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (AssignedTo) REFERENCES Users(UserID),
    FOREIGN KEY (CreatedBy) REFERENCES Users(UserID)
);

CREATE TABLE Comments (
    CommentID INT PRIMARY KEY AUTO_INCREMENT,
    TaskID INT NOT NULL,
    Comment TEXT,
    UserID INT,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (TaskID) REFERENCES Tasks(TaskID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

```python
from datetime import datetime
from app import db, app, bcrypt
from flask_login import UserMixin

class User(db.Model):
    __tablename__ = 'Users'

    id = db.Column('UserID', db.Integer, primary_key=True)
    username = db.Column('Username', db.String(50), nullable=False, unique=True)
    email = db.Column('Email', db.String(100), nullable=False, unique=True)
    password = db.Column('Password', db.String(100), nullable=False)
    created_at = db.Column('CreatedAt', db.DateTime, default=datetime.utcnow)

    # Relationships
    tasks_created = db.relationship('Task', backref='creator', lazy=True, foreign_keys='Task.created_by')
    tasks_assigned = db.relationship('Task', backref='assignee', lazy=True, foreign_keys='Task.assigned_to')
    comments = db.relationship('Comment', backref='user', lazy=True)

class Task(db.Model):
    __tablename__ = 'Tasks'

    id = db.Column('TaskID', db.Integer, primary_key=True)
    title = db.Column('Title', db.String(100), nullable=False)
    description = db.Column('Description', db.Text)
    status = db.Column('Status', db.Enum('Pending', 'In Progress', 'Completed'), nullable=False, default='Pending')
    priority = db.Column('Priority', db.Enum('Low', 'Medium', 'High'), nullable=False, default='Medium')
    category = db.Column('Category', db.String(100))
    assigned_to = db.Column('AssignedTo', db.Integer, db.ForeignKey('Users.UserID'))
    created_by = db.Column('CreatedBy', db.Integer, db.ForeignKey('Users.UserID'), nullable=False)
    due_date = db.Column('DueDate', db.Date)
    created_at = db.Column('CreatedAt', db.DateTime, default=datetime.utcnow)

    comments = db.relationship('Comment', backref='task', lazy=True)

class Comment(db.Model):
    __tablename__ = 'Comments'

    id = db.Column('CommentID', db.Integer, primary_key=True)
    task_id = db.Column('TaskID', db.Integer, db.ForeignKey('Tasks.TaskID'), nullable=False)
    comment_text = db.Column('Comment', db.Text)
    user_id = db.Column('UserID', db.Integer, db.ForeignKey('Users.UserID'))
    created_at = db.Column('CreatedAt', db.DateTime, default=datetime.utcnow)


class Login(db.Model, UserMixin):
    __tablename__ = 'login'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(255), unique=True, nullable=False)
    email = db.Column(db.String(255))
    password_hash = db.Column(db.String(255), nullable=False)

    def set_password(self, password):
        self.password_hash = bcrypt.generate_password_hash(password)

    def check_password(self, password):
        return bcrypt.check_password_hash(self.password_hash, password)

    def __repr__(self):
        return f'<Login {self.username}>'


with app.app_context():
    db.create_all()
```

# Backend - Routes

```python
@app.route('/')
def home():
    if current_user.is_authenticated:
        return redirect(url_for('dashboard'))
    return render_template('index.html')


@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = Login.query.filter_by(username=username).first()

        if user and user.check_password(password):
            login_user(user)
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username or password')
    if current_user.is_authenticated:
        return redirect(url_for('dashboard'))
    return render_template('login.html')

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']

        existing_user = Login.query.filter_by(username=username).first()
        existing_email = Login.query.filter_by(email=email).first()
        if existing_user is None and existing_email is None:
            new_user = Login(username=username, email=email)
            new_user.set_password(password)
            db.session.add(new_user)
            db.session.commit()
            return redirect(url_for('login'))
        elif existing_user is not None:
            flash('Username already exists')
        elif existing_email is not None:
            flash('Email already exists')
        else:
            flash('Something went wrong')
    if current_user.is_authenticated:
        return redirect(url_for('dashboard'))
    return render_template('signup.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('home'))
```

```python
@app.route('/dashboard', methods=['GET', 'POST'])
@login_required
def dashboard():
    user_filter = request.args.get('user', 'all')
    priority_filter = request.args.get('priority', 'all')

    query = Task.query.join(User, Task.assigned_to == User.id).add_columns(
        Task.id, Task.title, Task.description, Task.status, Task.priority, Task.category, Task.assigned_to, Task.due_date,
        User.username.label('assigned_username')
    )

    if user_filter != 'all':
        query = query.filter(Task.assigned_to == user_filter)
    if priority_filter != 'all':
        query = query.filter(Task.priority == priority_filter)

    tasks = query.all()
    users = User.query.all()
    #get the username for user_filter if it is not all, else set it to all
    if user_filter != 'all':
        user = User.query.get_or_404(user_filter).username
    else:
        user = 'all'
    return render_template('dashboard.html', tasks=tasks, users=users,user_filter=user, priority_filter=priority_filter)


@app.route('/users', methods=['GET', 'POST'])
@login_required
def users():
    users = User.query.all()
    return render_template('users.html', users=users)

@app.route('/create_user', methods=['GET', 'POST'])
@login_required
def create_user():
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')

        # Check if user already exists
        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            flash('Email already in use.', 'danger')
            return redirect(url_for('create_user'))

        # Create new user
        new_user = User(username=username, email=email, password=bcrypt.generate_password_hash(password).decode('utf-8'))
        db.session.add(new_user)
        db.session.commit()

        flash('User created successfully!', 'success')
        return redirect(url_for('users'))

    return render_template('create_user.html')
```

# Backend - Routes

```python
app.route('/delete_user/<int:user_id>', methods=['POST'])
@login_required
def delete_user(user_id):
    try:
        user = User.query.get_or_404(user_id)
        db.session.delete(user)
        db.session.commit()
        flash('User deleted successfully!', 'success')
    except SQLAlchemyError as e:
        flash('Error deleting user!', 'danger')
    return redirect(url_for('users'))


@app.route('/create_task', methods=['GET', 'POST'])
@login_required
def create_task():
    users = User.query.all()

    if request.method == 'POST':
        title = request.form.get('title')
        description = request.form.get('description')
        status = request.form.get('status')
        priority = request.form.get('priority')
        category = request.form.get('category')
        assigned_to = request.form.get('assignedTo')
        due_date = request.form.get('dueDate')
        created_by = request.form.get('createdBy')
        new_task = Task(
            title=title, description=description, status=status, priority=priority,
            category=category, assigned_to=assigned_to, created_by=created_by,
            due_date=due_date
        )
        db.session.add(new_task)
        db.session.commit()

        flash('Task created successfully!', 'success')
        return redirect(url_for('dashboard'))

    return render_template('create_task.html', users=users)


@app.route('/task_details/<int:task_id>')
@login_required
def task_details(task_id):
    query = Task.query.join(User, Task.assigned_to == User.id).add_columns(
        Task.id, Task.title, Task.description, Task.status, Task.priority, Task.category, Task.assigned_to, Task.due_date,
        User.username.label('assigned_username')
    )
    task = query.filter(Task.id == task_id).first()
    comments = Comment.query.filter_by(task_id=task_id).all()
    users = User.query.all()
    return render_template('task_details.html', task=task, comments=comments,users=users)
```

```python
@app.route('/edit_task/<int:task_id>', methods=['GET', 'POST'])
@login_required
def edit_task(task_id):
    task = Task.query.get_or_404(task_id)
    users = User.query.all()
    if request.method == 'POST':
        try:
            task.title = request.form.get('title')
            task.description = request.form.get('description')
            task.priority = request.form.get('priority')
            task.category = request.form.get('category')
            task.assigned_to = request.form.get('assignedTo')
            db.session.commit()
            flash('Task updated successfully!', 'success')
        except SQLAlchemyError as e:
            flash('Error updating task!', 'danger')
        return redirect(url_for('task_details', task_id=task_id))
    return render_template('edit_task.html', task=task, users=users)


@app.route('/add_comment/<int:task_id>', methods=['POST'])
@login_required
def add_comment(task_id):
    comment_text = request.form.get('comment')
    user_id = request.form.get('user_id')
    new_comment = Comment(task_id=task_id, comment_text=comment_text, user_id=user_id, created_at=datetime.now())
    db.session.add(new_comment)
    db.session.commit()
    return redirect(url_for('task_details', task_id=task_id))


@app.route('/delete_comment/<int:comment_id>', methods=['POST', 'GET'])
@login_required
def delete_comment(comment_id):
    try:
        comment = Comment.query.get_or_404(comment_id)
        db.session.delete(comment)
        db.session.commit()
        flash('Comment deleted successfully!', 'success')
    except SQLAlchemyError as e:
        flash('Error deleting comment!', 'danger')
    return redirect(url_for('dashboard'))


@app.route('/delete_task/<int:task_id>', methods=['POST', 'GET'])
@login_required
def delete_task(task_id):
    try:
        task = Task.query.get_or_404(task_id)
        print(task)
        db.session.delete(task)
        db.session.commit()
        flash('Task deleted successfully!', 'success')
    except SQLAlchemyError as e:
        flash('Error deleting task! Delete Comments First', 'danger')
    return redirect(url_for('dashboard'))

```