Казанников Андрей Андреевич БИВТ-23-6

Микросервис постинга и создания задач.

1. ЦЕЛЬ РАБОТЫ:

Целью данной работы является изучение и проектирование информационных систем с использованием диаграмм классов и диаграмм действий. Особое внимание уделяется пониманию ролей пользователей в системе и их возможностей.

2. Кратко по основным заданиям.

1. Создать диаграмму классов для системы управления задачами, включающей следующие сущности: пользователь (Users), задача (MyTaskModel), категория задач (TaskCategory), уведомления (UserNotification), и менеджеры для каждой из этих сущностей.
2. Создать диаграмму действий для пользователей системы с ролями "Админ" и "Обычный пользователь". Определить, какие действия могут выполнять эти роли.
3. Описать и реализовать основные методы для классов менеджеров.
4. Описать диаграмму постов и их особенностей

3. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ (ОПЦИОНАЛЬНО).

В теоретической части будет рассмотрена структура информационных систем и методология проектирования с использованием UML диаграмм. Диаграммы классов и действий являются основными инструментами для визуального представления структуры системы и ее поведения.

Диаграмма классов представляет собой статическую структуру системы, показывая классы, их атрибуты, методы и отношения между классами.

Диаграмма действий описывает динамическое поведение системы, показывая последовательность действий, выполняемых различными элементами системы.

4. БЛОК-СХЕМА

Диаграмма классов

```
classDiagram
    class Users {
```

```
        +Integer id
        +String username
        +String email
        +String password_hash
        +DateTime created_at
        +Boolean is_active
        +String role
        +set_password(password: String)
        +check_password(password: String) Boolean
    }

    class MyTaskModel {
        +Integer id
        +String task
        +DateTime added_time
        +Integer users_id
        +Integer category_id
        +__unicode__()
    }

    class TaskCategory {
        +Integer id
        +String name
        +__unicode__()
    }

    class UserNotification {
        +Integer id
        +String message
        +Boolean is_read
        +Integer user_id
        +DateTime created_time
        +__unicode__()
    }

    class MyTaskManager {
        +add_task(task_description: String, user_id: Integer,
category_id: Integer) MyTaskModel
        +update_task(task_id: Integer, new_description: String)
MyTaskModel
        +delete_task(task_id: Integer) Boolean
        +get_task(task_id: Integer) MyTaskModel
```

```
        +get_all_tasks_for_user(user_id: Integer) List~MyTaskModel~
        +search_tasks(search_term: String) List~MyTaskModel~
    }

    class TaskCategoryManager {
        +add_category(name: String) TaskCategory
        +get_all_categories() List~TaskCategory~
    }

    class UserNotificationManager {
        +add_notification(message: String, user_id: Integer)
UserNotification
        +mark_as_read(notification_id: Integer) UserNotification
        +get_unread_notifications(user_id: Integer)
List~UserNotification~
    }

    Users "1" o-- "0..*" MyTaskModel : Aggregation
    Users "1" o-- "0..*" UserNotification : Aggregation
    TaskCategory "1" o-- "0..*" MyTaskModel : Aggregation
    MyTaskManager *-- MyTaskModel : Composition
    TaskCategoryManager *-- TaskCategory : Composition
    UserNotificationManager *-- UserNotification : Composition
```

Диаграмма действий

```
flowchart TD
    A[Start] --> B[Choose Role]
    B --> C1[Admin]
    B --> C2[User]

    %% Admin Actions
    C1 --> D1[Manage Users]
    C1 --> D2[Manage Tasks]
    C1 --> D3[Manage Categories]
    C1 --> D4[Manage Notifications]
    C1 --> D5[Manage Posts]

    D1 --> E1[Create User]
    D1 --> E2[Edit User]
    D1 --> E3[Delete User]
```

```
D2 --> F1[Create Task]
D2 --> F2[Edit Task]
D2 --> F3[Delete Task]

D3 --> G1[Create Category]
D3 --> G2[Edit Category]
D3 --> G3[Delete Category]

D4 --> H1[View All Notifications]
D4 --> H2[Manage All Notifications]

D5 --> I1[Delete Post]

%% User Actions
C2 --> J1[Manage Own Tasks]
C2 --> J2[View Own Notifications]
C2 --> J3[Manage Posts]

J1 --> K1[Create Own Task]
J1 --> K2[Edit Own Task]
J1 --> K3[Delete Own Task]
J1 --> K4[View Own Task]

J2 --> L1[View Own Notifications]
J2 --> L2[Mark Notifications as Read]

J3 --> M1[Create Post]
J3 --> M2[View Own Posts]

%% End States
E1 --> Z[End]
E2 --> Z
E3 --> Z
F1 --> Z
F2 --> Z
F3 --> Z
G1 --> Z
G2 --> Z
G3 --> Z
H1 --> Z
H2 --> Z
```

```
    I1 --> Z
    K1 --> Z
    K2 --> Z
    K3 --> Z
    K4 --> Z
    L1 --> Z
    L2 --> Z
    M1 --> Z
    M2 --> Z
```

5. ЛИСТИНГ ПРОГРАММЫ

Models

```
from sqlalchemy import Column, Integer, String, DateTime, Boolean,
ForeignKey
from sqlalchemy.orm import relationship
from flask_login import UserMixin
from werkzeug.security import generate_password_hash,
check_password_hash
from ..extensions import db
from ..utils import get_current_time

class Users(db.Model, UserMixin):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    username = Column(String(64), unique=True, nullable=False)
    email = Column(String(120), unique=True, nullable=False)
    password_hash = Column(String(128))
    created_at = Column(DateTime, default=get_current_time)
    is_active = Column(Boolean, default=True)
    role = Column(String(64), default='user')

    tasks = relationship("MyTaskModel", backref="user",
lazy='dynamic')
    notifications = relationship("UserNotification", backref="user",
lazy='dynamic')

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

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

    def __unicode__(self):
        return f'User ID: {self.id}, Username: {self.username}'


class MyTaskModel(db.Model):
    __tablename__ = 'mytask_model'

    id = Column(Integer, primary_key=True)
    task = Column(String(2048))
    added_time = Column(DateTime, default=get_current_time)
    users_id = Column(Integer, ForeignKey("users.id"))
    category_id = Column(Integer, ForeignKey("task_category.id"))

    def __unicode__(self):
        return f'ID: {self.id}, Task: {self.task}'


class TaskCategory(db.Model):
    __tablename__ = 'task_category'

    id = Column(Integer, primary_key=True)
    name = Column(String(255), unique=True)
    tasks = relationship("MyTaskModel", backref="category")

    def __unicode__(self):
        return self.name


class UserNotification(db.Model):
    __tablename__ = 'user_notification'

    id = Column(Integer, primary_key=True)
    message = Column(String(1024))
    is_read = Column(Boolean, default=False)
    user_id = Column(Integer, ForeignKey("users.id"))
    created_time = Column(DateTime, default=get_current_time)

    def __unicode__(self):
```

```
        return f'ID: {self.id}, Message: {self.message}'
```

Managers

```
from sqlalchemy.exc import SQLAlchemyError

class MyTaskManager:
    def __init__(self, db_session):
        self.db_session = db_session

    def add_task(self, task_description, user_id, category_id=None):
        new_task = MyTaskModel(task=task_description,
users_id=user_id, category_id=category_id)
        self.db_session.add(new_task)
        try:
            self.db_session.commit()
            return new_task
        except SQLAlchemyError as e:
            self.db_session.rollback()
            print(f"Error adding task: {e}")
            return None

    def update_task(self, task_id, new_description):
        task = self.db_session.query(MyTaskModel).get(task_id)
        if task:
            task.task = new_description
            try:
                self.db_session.commit()
                return task
            except SQLAlchemyError as e:
                self.db_session.rollback()
                print(f"Error updating task: {e}")
                return None
        return None

    def delete_task(self, task_id):
        task = self.db_session.query(MyTaskModel).get(task_id)
        if task:
            self.db_session.delete(task)
            try:
                self.db_session.commit()
```

```
                return True
            except SQLAlchemyError as e:
                self.db_session.rollback()
                print(f"Error deleting task: {e}")
                return False
        return False

    def get_task(self, task_id):
        return self.db_session.query(MyTaskModel).get(task_id)

    def get_all_tasks_for_user(self, user_id):
        return
self.db_session.query(MyTaskModel).filter_by(users_id=user_id).all()

    def search_tasks(self, search_term):
        return
self.db_session.query(MyTaskModel).filter(MyTaskModel.task.like(f"%{
search_term}%")).all()


class TaskCategoryManager:
    def __init__(self, db_session):
        self.db_session = db_session

    def add_category(self, name):
        new_category = TaskCategory(name=name)
        self.db_session.add(new_category)
        try:
            self.db_session.commit()
            return new_category
        except SQLAlchemyError as e:
            self.db_session.rollback()
            print(f"Error adding category: {e}")
            return None

    def get_all_categories(self):
        return self.db_session.query(TaskCategory).all()


class UserNotificationManager:
    def __init__(self, db_session):
        self.db_session = db_session
```

```
    def add_notification(self, message, user_id):
        new_notification = UserNotification(message=message,
user_id=user_id)
        self.db_session.add(new_notification)
        try:
            self.db_session.commit()
            return new_notification
        except SQLAlchemyError as e:
            self.db_session.rollback()
            print(f"Error adding notification: {e}")
            return None

    def mark_as_read(self, notification_id):
        notification =
self.db_session.query(UserNotification).get(notification_id)
        if notification:
            notification.is_read = True
            try:
                self.db_session.commit()
                return notification
            except SQLAlchemyError as e:
                self.db_session.rollback()
                print(f"Error marking notification as read: {e}")
                return None
        return None

    def get_unread_notifications(self, user_id):
        return
self.db_session.query(UserNotification).filter_by(user_id=user_id,
is_read=False).all()
```

7. ВЫВОД

В данной работе была разработана информационная система управления задачами с использованием диаграмм классов и действий. Система поддерживает две роли пользователей: Админ и Обычный пользователь. Администраторы могут управлять пользователями, задачами, категориями и уведомлениями, а также удалять посты. Обычные пользователи могут создавать, редактировать и удалять свои задачи, просматривать свои уведомления и создавать посты. Разработанные диаграммы и классы облегчают понимание

структуры и поведения системы, а также обеспечивают основу для дальнейшего развития и модификации.