**Министерство транспорта Российской Федерации**

**Федеральное государственное автономное образовательное  
учреждение высшего образования  
«РУТ (МИИТ)»**

**Институт транспортной техники и систем управления**

**Кафедра «Управление и защита информации»**

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

**«Основы построения защищенных баз данных»**

**на тему  
«Проектирование базы данных»**

Выполнила: ст. гр. ТКИ-541,  
Пирогова К. Д.  
Проверил: доц., к.т.н.,

Васильева М. А.

Москва 2023

Оглавление

[ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ 6](#_Toc149071925)

[ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ 6](#_Toc149071926)

[ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ 6](#_Toc149071927)

[UML - ДИАГРАММА КЛАССОВ 7](#_Toc149071928)

[СТРУКТУРА ПРОЕКТА 8](#_Toc149071929)

[КОД ПРОЕКТА 8](#_Toc149071930)

[models/base.py 8](#_Toc149071931)

[models/ employee\_equipment.py 8](#_Toc149071932)

[models/employee.py 9](#_Toc149071933)

[models/equipment.py 11](#_Toc149071934)

[models/position.py 12](#_Toc149071935)

[models/time\_track.py 13](#_Toc149071936)

[ПРОХОЖДЕНИЕ ТЕСТОВ 14](#_Toc149071937)

[tests/ test\_employee\_equipment\_model.py 14](#_Toc149071938)

[tests/ test\_employee\_model.py 17](#_Toc149071939)

[tests/ test\_equipment\_model.py 20](#_Toc149071940)

[tests/ test\_mapping.py 23](#_Toc149071941)

[tests/ test\_position\_model.py 25](#_Toc149071942)

[tests/ test\_time\_track\_model.py 27](#_Toc149071943)

[ERD - ДИАГРАММА 30](#_Toc149071944)

[СПИСОК ЛИТЕРАТУРЫ 32](#_Toc149071945)

# **ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

Разработать систему учета пользователей, размещенных ими рецептов и оставленных отзывов.

# **ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ**

Научиться эффективно работать с базами данных, используя подход ORM (Object-Relational Mapping), для создания, модификации, запроса и управления данными в структурированной и объектно-ориентированной манере.

# **ИСПОЛЬЗУЕМЫЙ СТЕК ТЕХНОЛОГИЙ**

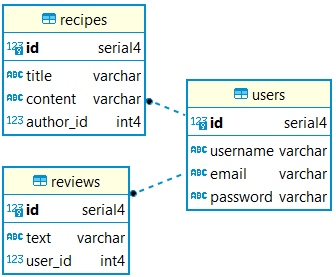
1. Python: основной язык программирования, на котором написан предоставленный код.

2. SQLAlchemy: ORM-библиотека для Python, которая позволяет взаимодействовать с базами данных на высоком уровне, используя объектно-ориентированный подход. С ее помощью можно создавать модели, связи между ними и формировать запросы к базе данных.

3. SQL: язык структурированных запросов, используемый для определения и управления данными в реляционных базах данных. В коде он не присутствует напрямую, но SQLAlchemy генерирует SQL-запросы на основе объектных моделей и методов.

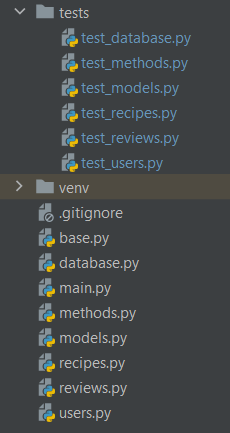
4. DBeaver: универсальный инструмент для работы с базами данных, который поддерживает множество различных СУБД.

# **UML - ДИАГРАММА КЛАССОВ**

**

*Рисунок 1 - UML-диаграмма классов*

# **СТРУКТУРА ПРОЕКТА**



*Рисунок 2 – Структура проекта*

# **КОД ПРОЕКТА**

**base.py**

from sqlalchemy.ext.declarative import declarative\_base  
Base = declarative\_base()

**database.py**

from sqlalchemy import create\_engine  
from sqlalchemy.orm import sessionmaker  
from base import Base  
db\_url = 'postgresql://postgres:1@localhost:5432/postgres'  
  
engine = create\_engine(db\_url)  
  
def create\_tables():  
 Base.metadata.create\_all(bind=engine)  
  
def create\_session():  
 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
 session = SessionLocal()

**main.py**

from database import engine, sessionmaker  
from methods import UserMethod, ReviewMethod, RecipeMethod  
from base import Base  
  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
session = SessionLocal()  
Base.metadata.create\_all(bind=engine)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 session = SessionLocal()  
  
 # Примеры использования методов  
 user\_method = UserMethod(session)  
 review\_method = ReviewMethod(session)  
 recipe\_method = RecipeMethod(session)  
  
new\_user = user\_method.create\_user(username="john", email="john@example.com", password="password123")  
print("Created User:", new\_user)  
  
# Получение пользователя по ID  
retrieved\_user = user\_method.get\_user(user\_id=new\_user.id)  
print("Retrieved User:", retrieved\_user)  
  
# Получение всех пользователей  
all\_users = user\_method.get\_all\_users()  
print("All Users:", all\_users)  
  
# Обновление пользователя  
updated\_user = user\_method.update\_user(user\_id=new\_user.id, email="new\_email@example.com")  
print("Updated User:", updated\_user)  
  
# Удаление пользователя  
#user\_method.delete\_user(user\_id=new\_user.id)  
#print("User deleted")  
  
# Создание нового отзыва  
new\_review = review\_method.create\_review(text="Great product!", user\_id=1)  
print("Created Review:", new\_review)  
  
# Получение отзыва по ID  
retrieved\_review = review\_method.get\_review(review\_id=new\_review.id)  
print("Retrieved Review:", retrieved\_review)  
  
# Получение всех отзывов  
all\_reviews = review\_method.get\_all\_reviews()  
print("All Reviews:", all\_reviews)  
  
# Обновление отзыва  
updated\_review = review\_method.update\_review(review\_id=new\_review.id, text="Updated review")  
print("Updated Review:", updated\_review)  
  
# Удаление отзыва  
#review\_method.delete\_review(review\_id=new\_review.id)  
#print("Review deleted")  
  
# Создание нового рецепта  
new\_recipe = recipe\_method.create\_recipe(title="Chocolate Cake", content="Best chocolate cake recipe!", author\_id=1)  
print("Created Recipe:", new\_recipe)  
  
# Получение рецепта по ID  
retrieved\_recipe = recipe\_method.get\_recipe(recipe\_id=new\_recipe.id)  
print("Retrieved Recipe:", retrieved\_recipe)  
  
# Получение всех рецептов  
all\_recipes = recipe\_method.get\_all\_recipes()  
print("All Recipes:", all\_recipes)  
  
# Обновление рецепта  
updated\_recipe = recipe\_method.update\_recipe(recipe\_id=new\_recipe.id, title="Updated chocolate cake recipe")  
print("Updated Recipe:", updated\_recipe)  
  
# Удаление рецепта  
#recipe\_method.delete\_recipe(recipe\_id=new\_recipe.id)  
#print("Recipe deleted")  
  
# Закрытие сессии  
session.close()

**methods.py**

from models import User, Review, Recipe  
from database import engine, sessionmaker  
  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
session = SessionLocal()  
class BaseMethod:  
 def \_\_init\_\_(self, session):  
 self.session = session  
  
 def create(self, model, \*\*kwargs):  
 instance = model(\*\*kwargs)  
 self.session.add(instance)  
 self.session.commit()  
 self.session.refresh(instance)  
 return instance  
  
 def get(self, model, id):  
 return self.session.query(model).filter\_by(id=id).first()  
  
 def get\_all(self, model):  
 return self.session.query(model).all()  
  
 def update(self, instance, \*\*kwargs):  
 for key, value in kwargs.items():  
 setattr(instance, key, value)  
 self.session.commit()  
 self.session.refresh(instance)  
 return instance  
  
 def delete(self, instance):  
 self.session.delete(instance)  
 self.session.commit()  
  
class UserMethod(BaseMethod):  
 def create\_user(self, username: str, email: str, password: str):  
 user = User(username=username, email=email, password=password)  
 self.session.add(user)  
 self.session.commit()  
 return user  
  
 def get\_user(self, user\_id: int):  
 return self.get(User, user\_id)  
  
 def get\_user\_by\_username(self, username: str):  
 return self.session.query(User).filter\_by(username=username).first()  
  
  
 def get\_all\_users(self):  
 return self.get\_all(User)  
  
 def update\_user(self, user\_id: int, \*\*kwargs):  
 user = self.get\_user(user\_id)  
 return self.update(user, \*\*kwargs)  
  
 def delete\_user(self, user\_id: int):  
 user = self.get\_user(user\_id)  
 self.delete(user)  
  
class ReviewMethod(BaseMethod):  
 def create\_review(self, text: str, user\_id: int):  
 return self.create(Review, text=text, user\_id=user\_id)  
  
 def get\_review(self, review\_id):  
 return self.get(Review, review\_id)  
  
 def get\_all\_reviews(self):  
 return self.get\_all(Review)  
  
 def update\_review(self, review\_id, \*\*kwargs):  
 review = self.get\_review(review\_id)  
 return self.update(review, \*\*kwargs)  
  
 def delete\_review(self, review\_id):  
 review = self.get\_review(review\_id)  
 self.delete(review)  
  
class RecipeMethod(BaseMethod):  
 def create\_recipe(self, title: str, content: str, author\_id: int):  
 return self.create(Recipe, title=title, content=content, author\_id=author\_id)  
  
 def get\_recipe(self, recipe\_id):  
 return self.get(Recipe, recipe\_id)  
  
 def get\_all\_recipes(self):  
 return self.get\_all(Recipe)  
  
 def update\_recipe(self, recipe\_id, \*\*kwargs):  
 recipe = self.get\_recipe(recipe\_id)  
 return self.update(recipe, \*\*kwargs)  
  
 def delete\_recipe(self, recipe\_id):  
 recipe = self.get\_recipe(recipe\_id)  
 self.delete(recipe)

**models.py**

from sqlalchemy import Column, Integer, String, ForeignKey  
from sqlalchemy.orm import relationship  
from database import Base  
  
class User(Base):  
 \_\_tablename\_\_ = "users"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 username = Column(String, index=True, nullable=False)  
 email = Column(String, unique=True, index=True, nullable=False)  
 password = Column(String, index=True, nullable=False)  
 recipes = relationship("Recipe", back\_populates="author")  
 reviews = relationship("Review", back\_populates="author")  
  
class Review(Base):  
 \_\_tablename\_\_ = "reviews"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 text = Column(String, nullable=False)  
 user\_id = Column(Integer, ForeignKey("users.id"), nullable=False)  
 author = relationship("User", back\_populates="reviews")  
  
class Recipe(Base):  
 \_\_tablename\_\_ = "recipes"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 title = Column(String, nullable=False)  
 content = Column(String, nullable=False)  
 author\_id = Column(Integer, ForeignKey("users.id"), nullable=False)  
 author = relationship("User", back\_populates="recipes")

**recipes.py**

from sqlalchemy import Column, Integer, String, ForeignKey, Date, Text  
from sqlalchemy.orm import relationship  
from base import Base  
  
class Recipe(Base):  
 \_\_tablename\_\_ = 'recipes'  
  
 id = Column(Integer, primary\_key=True, index=True)  
 title = Column(String, nullable=False)  
 content = Column(Text, nullable=False)  
 author\_id = Column(Integer, ForeignKey('users.id'), nullable=False)  
 author = relationship('User', back\_populates='recipes')  
  
 def \_\_init\_\_(self, title: str, content: str, author\_id: int):  
 self.title = title  
 self.content = content  
 self.author\_id = author\_id

**reviews.py**

from sqlalchemy import Column, Integer, String, ForeignKey, Date, Text  
from sqlalchemy.orm import relationship  
from base import Base  
class Review(Base):  
 \_\_tablename\_\_ = 'reviews'  
  
 id = Column(Integer, primary\_key=True, index=True)  
 text = Column(Text, nullable=False)  
 user\_id = Column(Integer, ForeignKey('users.id'), nullable=False)  
 user = relationship('User', back\_populates='reviews')  
  
 def \_\_init\_\_(self, text: str, user\_id: int):  
 self.text = text  
 self.user\_id = user\_id

**users.py**

from sqlalchemy import Column, Integer, String, ForeignKey, Text  
from sqlalchemy.orm import relationship  
from base import Base  
  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
 id = Column(Integer, primary\_key=True, index=True)  
 username = Column(String, unique=True, nullable=False)  
 email = Column(String, unique=True, nullable=False)  
 password = Column(Text, nullable=False)  
 reviews = relationship('Review', back\_populates='user')  
 recipes = relationship('Recipe', back\_populates='author')  
  
 def \_\_init\_\_(self, username: str, email: str, password: str):  
 self.username = username  
 self.email = email  
 self.password = password

**tests/test\_database.py**

import unittest  
from sqlalchemy import create\_engine, inspect  
from database import create\_tables, create\_session  
from sqlalchemy.orm import sessionmaker  
  
class TestDatabase(unittest.TestCase):  
 def test\_db\_connection(self):  
 db\_url = 'postgresql://postgres:1@localhost:5432/postgres'  
 engine = create\_engine(db\_url)  
 connection = engine.connect()  
 self.assertIsNotNone(connection)  
 connection.close()  
  
 def test\_create\_session(self):  
 db\_url = 'postgresql://postgres:1@localhost:5432/postgres'  
  
 engine = create\_engine(db\_url)  
  
 SessionLocal = sessionmaker(bind=engine)  
 session = SessionLocal()  
  
 self.assertIsNotNone(session)  
 session.close()  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**tests/test\_methods.py**

import unittest  
from sqlalchemy import create\_engine  
from sqlalchemy.orm import Session  
from models import User, Review, Recipe  
from methods import BaseMethod, UserMethod, ReviewMethod, RecipeMethod  
from base import Base  
  
class TestBaseMethod(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
 self.base\_method = BaseMethod(session=self.session)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_create(self):  
 # Test create method  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = self.base\_method.create(User, \*\*user\_data)  
 self.assertIsNotNone(user.id)  
  
 def test\_get(self):  
 # Test get method  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = self.base\_method.create(User, \*\*user\_data)  
 retrieved\_user = self.base\_method.get(User, user.id)  
 self.assertEqual(retrieved\_user.username, 'test\_user')  
  
 def test\_get\_all(self):  
 # Test get\_all method  
 user\_data\_1 = {'username': 'test\_user1', 'email': 'test1@example.com', 'password': 'test\_password1'}  
 user\_data\_2 = {'username': 'test\_user2', 'email': 'test2@example.com', 'password': 'test\_password2'}  
 self.base\_method.create(User, \*\*user\_data\_1)  
 self.base\_method.create(User, \*\*user\_data\_2)  
  
 all\_users = self.base\_method.get\_all(User)  
 self.assertEqual(len(all\_users), 2)  
  
 def test\_update(self):  
 # Test update method  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = self.base\_method.create(User, \*\*user\_data)  
  
 updated\_user = self.base\_method.update(user, username='updated\_user')  
 self.assertEqual(updated\_user.username, 'updated\_user')  
  
 def test\_delete(self):  
 # Test delete method  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = self.base\_method.create(User, \*\*user\_data)  
  
 self.base\_method.delete(user)  
 deleted\_user = self.base\_method.get(User, user.id)  
 self.assertIsNone(deleted\_user)  
  
class TestUserMethod(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
 self.user\_method = UserMethod(session=self.session)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_create\_user(self):  
 # Test create\_user method  
 user = self.user\_method.create\_user(username='test\_user', email='test@example.com', password='test\_password')  
 self.assertIsNotNone(user.id)  
  
 def test\_get\_user(self):  
 # Test get\_user method  
 user = self.user\_method.create\_user(username='test\_user', email='test@example.com', password='test\_password')  
 retrieved\_user = self.user\_method.get\_user(user.id)  
 self.assertEqual(retrieved\_user.username, 'test\_user')  
  
 def test\_get\_all\_users(self):  
 # Test get\_all\_users method  
 user\_data\_1 = {'username': 'test\_user1', 'email': 'test1@example.com', 'password': 'test\_password1'}  
 user\_data\_2 = {'username': 'test\_user2', 'email': 'test2@example.com', 'password': 'test\_password2'}  
 self.user\_method.create\_user(\*\*user\_data\_1)  
 self.user\_method.create\_user(\*\*user\_data\_2)  
  
 all\_users = self.user\_method.get\_all\_users()  
 self.assertEqual(len(all\_users), 2)  
  
 def test\_update\_user(self):  
 # Test update\_user method  
 user = self.user\_method.create\_user(username='test\_user', email='test@example.com', password='test\_password')  
  
 updated\_user = self.user\_method.update\_user(user.id, username='updated\_user')  
 self.assertEqual(updated\_user.username, 'updated\_user')  
  
 def test\_delete\_user(self):  
 # Test delete\_user method  
 user = self.user\_method.create\_user(username='test\_user', email='test@example.com', password='test\_password')  
  
 self.user\_method.delete\_user(user.id)  
 deleted\_user = self.user\_method.get\_user(user.id)  
 self.assertIsNone(deleted\_user)  
  
class TestReviewMethod(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
 self.review\_method = ReviewMethod(session=self.session)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_create\_review(self):  
 # Test create\_review method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 review = self.review\_method.create\_review(text='Test review text', user\_id=user.id)  
 self.assertIsNotNone(review.id)  
  
 def test\_get\_review(self):  
 # Test get\_review method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 review = self.review\_method.create\_review(text='Test review text', user\_id=user.id)  
 retrieved\_review = self.review\_method.get\_review(review.id)  
 self.assertEqual(retrieved\_review.text, 'Test review text')  
  
 def test\_get\_all\_reviews(self):  
 # Test get\_all\_reviews method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 review\_data\_1 = {'text': 'Review 1', 'user\_id': user.id}  
 review\_data\_2 = {'text': 'Review 2', 'user\_id': user.id}  
 self.review\_method.create\_review(\*\*review\_data\_1)  
 self.review\_method.create\_review(\*\*review\_data\_2)  
  
 all\_reviews = self.review\_method.get\_all\_reviews()  
 self.assertEqual(len(all\_reviews), 2)  
  
 def test\_update\_review(self):  
 # Test update\_review method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 review = self.review\_method.create\_review(text='Test review text', user\_id=user.id)  
  
 updated\_review = self.review\_method.update\_review(review.id, text='Updated review text')  
 self.assertEqual(updated\_review.text, 'Updated review text')  
  
 def test\_delete\_review(self):  
 # Test delete\_review method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 review = self.review\_method.create\_review(text='Test review text', user\_id=user.id)  
  
 self.review\_method.delete\_review(review.id)  
 deleted\_review = self.review\_method.get\_review(review.id)  
 self.assertIsNone(deleted\_review)  
  
class TestRecipeMethod(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
 self.recipe\_method = RecipeMethod(session=self.session)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_create\_recipe(self):  
 # Test create\_recipe method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 recipe = self.recipe\_method.create\_recipe(title='Test Recipe', content='Recipe content', author\_id=user.id)  
 self.assertIsNotNone(recipe.id)  
  
 def test\_get\_recipe(self):  
 # Test get\_recipe method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 recipe = self.recipe\_method.create\_recipe(title='Test Recipe', content='Recipe content', author\_id=user.id)  
 retrieved\_recipe = self.recipe\_method.get\_recipe(recipe.id)  
 self.assertEqual(retrieved\_recipe.title, 'Test Recipe')  
  
 def test\_get\_all\_recipes(self):  
 # Test get\_all\_recipes method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 recipe\_data\_1 = {'title': 'Recipe 1', 'content': 'Content 1', 'author\_id': user.id}  
 recipe\_data\_2 = {'title': 'Recipe 2', 'content': 'Content 2', 'author\_id': user.id}  
 self.recipe\_method.create\_recipe(\*\*recipe\_data\_1)  
 self.recipe\_method.create\_recipe(\*\*recipe\_data\_2)  
  
 all\_recipes = self.recipe\_method.get\_all\_recipes()  
 self.assertEqual(len(all\_recipes), 2)  
  
 def test\_update\_recipe(self):  
 # Test update\_recipe method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 recipe = self.recipe\_method.create\_recipe(title='Test Recipe', content='Recipe content', author\_id=user.id)  
  
 updated\_recipe = self.recipe\_method.update\_recipe(recipe.id, title='Updated Recipe')  
 self.assertEqual(updated\_recipe.title, 'Updated Recipe')  
  
 def test\_delete\_recipe(self):  
 # Test delete\_recipe method  
 user = User(username='test\_user', email='test@example.com', password='test\_password')  
 self.session.add(user)  
 self.session.commit()  
  
 recipe = self.recipe\_method.create\_recipe(title='Test Recipe', content='Recipe content', author\_id=user.id)  
  
 self.recipe\_method.delete\_recipe(recipe.id)  
 deleted\_recipe = self.recipe\_method.get\_recipe(recipe.id)  
 self.assertIsNone(deleted\_recipe)  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**tests/test\_models.py**

import unittest  
from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey  
from sqlalchemy.orm import Session, relationship  
from database import Base  
from models import User, Review, Recipe  
  
class TestModels(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_user\_creation(self):  
 # Test user creation and save to the database  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = User(\*\*user\_data)  
  
 # Add user to the session and commit to the database  
 self.session.add(user)  
 self.session.commit()  
  
 # Retrieve the user from the database and assert values  
 retrieved\_user = self.session.query(User).filter\_by(username='test\_user').first()  
 self.assertIsNotNone(retrieved\_user)  
 self.assertEqual(retrieved\_user.email, 'test@example.com')  
 self.assertEqual(retrieved\_user.password, 'test\_password')  
  
 def test\_review\_creation(self):  
 # Test review creation and save to the database  
 review\_data = {'text': 'Test review text', 'user\_id': 1}  
 review = Review(\*\*review\_data)  
  
 # Add review to the session and commit to the database  
 self.session.add(review)  
 self.session.commit()  
  
 # Retrieve the review from the database and assert values  
 retrieved\_review = self.session.query(Review).filter\_by(text='Test review text').first()  
 self.assertIsNotNone(retrieved\_review)  
 self.assertEqual(retrieved\_review.user\_id, 1)  
  
 def test\_recipe\_creation(self):  
 # Test recipe creation and save to the database  
 recipe\_data = {'title': 'Test Recipe', 'content': 'Recipe content', 'author\_id': 1}  
 recipe = Recipe(\*\*recipe\_data)  
  
 # Add recipe to the session and commit to the database  
 self.session.add(recipe)  
 self.session.commit()  
  
 # Retrieve the recipe from the database and assert values  
 retrieved\_recipe = self.session.query(Recipe).filter\_by(title='Test Recipe').first()  
 self.assertIsNotNone(retrieved\_recipe)  
 self.assertEqual(retrieved\_recipe.content, 'Recipe content')  
 self.assertEqual(retrieved\_recipe.author\_id, 1)  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

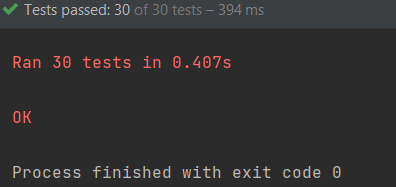
**tests/test\_recipes.py**

import unittest  
from sqlalchemy import create\_engine  
from sqlalchemy.orm import Session  
from base import Base  
from models import Recipe,User  
  
class TestRecipe(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_recipe\_creation(self):  
 # Test recipe creation and save to the database  
 recipe\_data = {'title': 'Test Recipe', 'content': 'Recipe content', 'author\_id': 1}  
 recipe = Recipe(\*\*recipe\_data)  
  
 # Add recipe to the session and commit to the database  
 self.session.add(recipe)  
 self.session.commit()  
  
 # Retrieve the recipe from the database and assert values  
 retrieved\_recipe = self.session.query(Recipe).filter\_by(title='Test Recipe').first()  
 self.assertIsNotNone(retrieved\_recipe)  
 self.assertEqual(retrieved\_recipe.title, 'Test Recipe')  
 self.assertEqual(retrieved\_recipe.content, 'Recipe content')  
 self.assertEqual(retrieved\_recipe.author\_id, 1)  
  
 def test\_recipe\_relationship\_with\_author(self):  
 # Test the relationship between Recipe and User  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = User(\*\*user\_data)  
 self.session.add(user)  
 self.session.commit()  
  
 recipe\_data = {'title': 'Test Recipe', 'content': 'Recipe content', 'author\_id': user.id}  
 recipe = Recipe(\*\*recipe\_data)  
  
 self.session.add(recipe)  
 self.session.commit()  
  
 # Retrieve the user with recipes and assert the relationship  
 retrieved\_user = self.session.query(User).filter\_by(username='test\_user').first()  
 self.assertIsNotNone(retrieved\_user)  
 self.assertEqual(len(retrieved\_user.recipes), 1)  
 self.assertEqual(retrieved\_user.recipes[0].title, 'Test Recipe')  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()  
**tests/test\_reviews.py**

import unittest  
from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey  
from sqlalchemy.orm import Session, relationship  
from base import Base  
from models import Review, User  
  
class TestReview(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_review\_creation(self):  
 # Test review creation and save to the database  
 review\_data = {'text': 'Test review text', 'user\_id': 1}  
 review = Review(\*\*review\_data)  
  
 # Add review to the session and commit to the database  
 self.session.add(review)  
 self.session.commit()  
  
 # Retrieve the review from the database and assert values  
 retrieved\_review = self.session.query(Review).filter\_by(text='Test review text').first()  
 self.assertIsNotNone(retrieved\_review)  
 self.assertEqual(retrieved\_review.text, 'Test review text')  
 self.assertEqual(retrieved\_review.user\_id, 1)  
  
 def test\_review\_relationship\_with\_user(self):  
 # Test the relationship between Review and User  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = User(\*\*user\_data)  
 self.session.add(user)  
 self.session.commit()  
  
 review\_data = {'text': 'Test review text', 'user\_id': user.id}  
 review = Review(\*\*review\_data)  
  
 self.session.add(review)  
 self.session.commit()  
  
 # Retrieve the user with reviews and assert the relationship  
 retrieved\_user = self.session.query(User).filter\_by(username='test\_user').first()  
 self.assertIsNotNone(retrieved\_user)  
 self.assertEqual(len(retrieved\_user.reviews), 1)  
 self.assertEqual(retrieved\_user.reviews[0].text, 'Test review text')  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

**tests/test\_users.py**

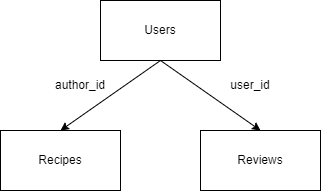
import unittest  
from sqlalchemy import create\_engine  
from sqlalchemy.orm import Session  
from base import Base  
from models import User  
  
class TestUser(unittest.TestCase):  
 def setUp(self):  
 self.engine = create\_engine('sqlite:///:memory:')  
 Base.metadata.create\_all(bind=self.engine)  
 self.session = Session(bind=self.engine)  
  
 def tearDown(self):  
 Base.metadata.drop\_all(bind=self.engine)  
 self.session.close()  
  
 def test\_user\_creation(self):  
 # Test user creation and save to the database  
 user\_data = {'username': 'test\_user', 'email': 'test@example.com', 'password': 'test\_password'}  
 user = User(\*\*user\_data)  
  
 # Add user to the session and commit to the database  
 self.session.add(user)  
 self.session.commit()  
  
 # Retrieve the user from the database and assert values  
 retrieved\_user = self.session.query(User).filter\_by(username='test\_user').first()  
 self.assertIsNotNone(retrieved\_user)  
 self.assertEqual(retrieved\_user.username, 'test\_user')  
 self.assertEqual(retrieved\_user.email, 'test@example.com')  
 self.assertEqual(retrieved\_user.password, 'test\_password')  
  
 if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()



*Рисунок 3 – Результат прохождения тестов*

# **ERD - ДИАГРАММА**

Диаграмма ERD (Entity-Relationship Diagram) представляет собой графическое изображение структуры данных, используемое для высокоуровневого проектирования базы данных. Она иллюстрирует отношения между различными сущностями или объектами в базе данных.

**

*Рисунок 4 – ERD*

# **СПИСОК ЛИТЕРАТУРЫ**

* + - 1. Васильева М.А., Хобта Д.О., Фильтрация набора данных. Рекомендации по выполнению работы и перечень типовых заданий: Учебно-методическое пособие. Издание второе, исправленное и дополненное–М.:РУТ(МИИТ). 2023.–105с.
      2. Васильева М.А., Меркулов Д.А. Группировка и обобщение данных. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 46–с.
      3. Васильева М.А., Ракинцев Н.А. Соединение данных из множества таблиц. Рекомендации по выполнению работы и перечень типовых заданий. Учебно-методическое пособие. М.:РУТ(МИИТ), 2023. 63–с.
      4. Балакина Е.П., Васильева М.А., Филипченко К.М. Информационное обеспечение систем управления. Методические указания к курсовому проектированию. Учебно-методическое пособие. Издание второе, исправленное и дополненное, 2023.102–с.