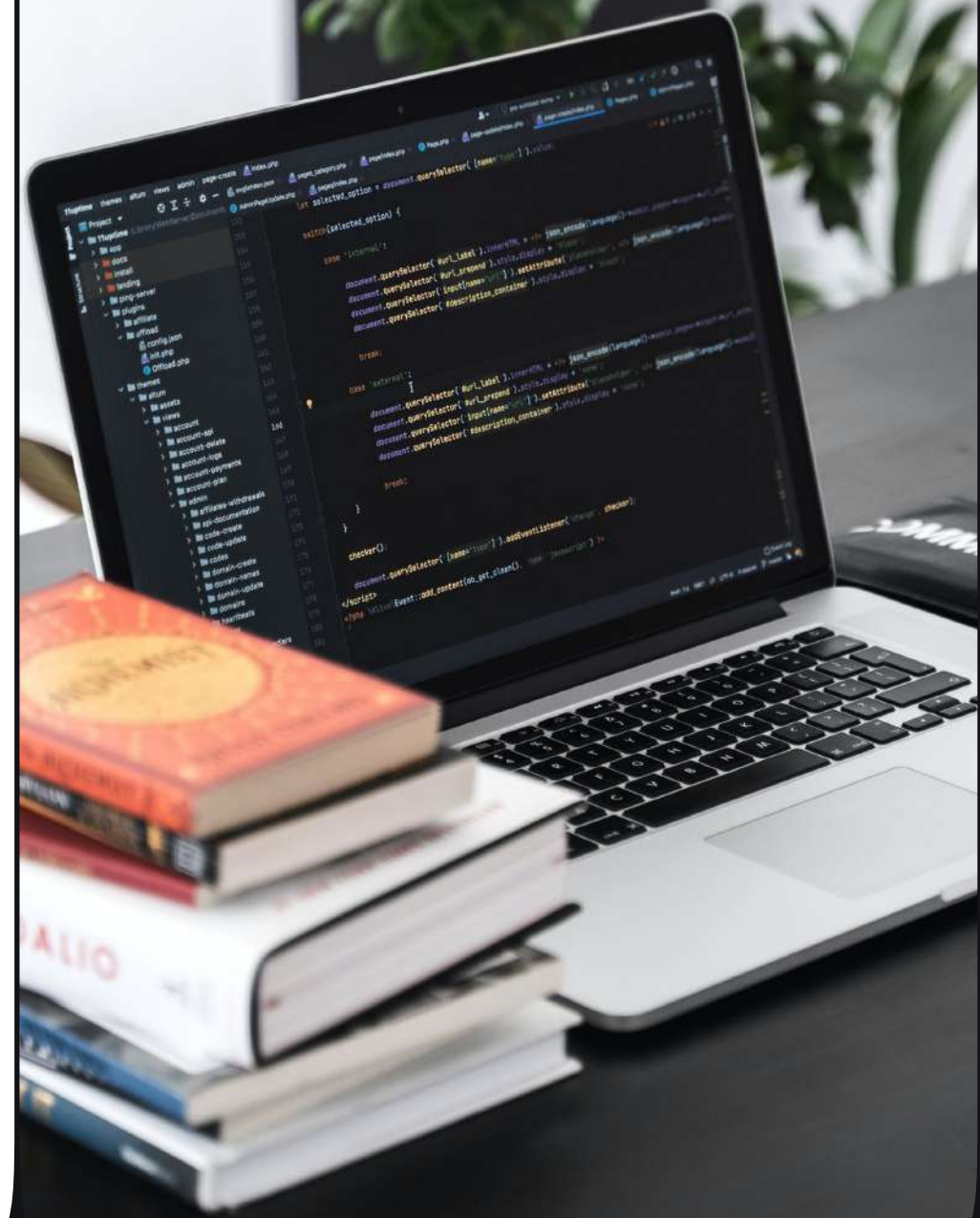


Модуль 4 Занятие 6

Создание баз данных во Flask



ORM

SQLAlchemy

Модели

Поля

+

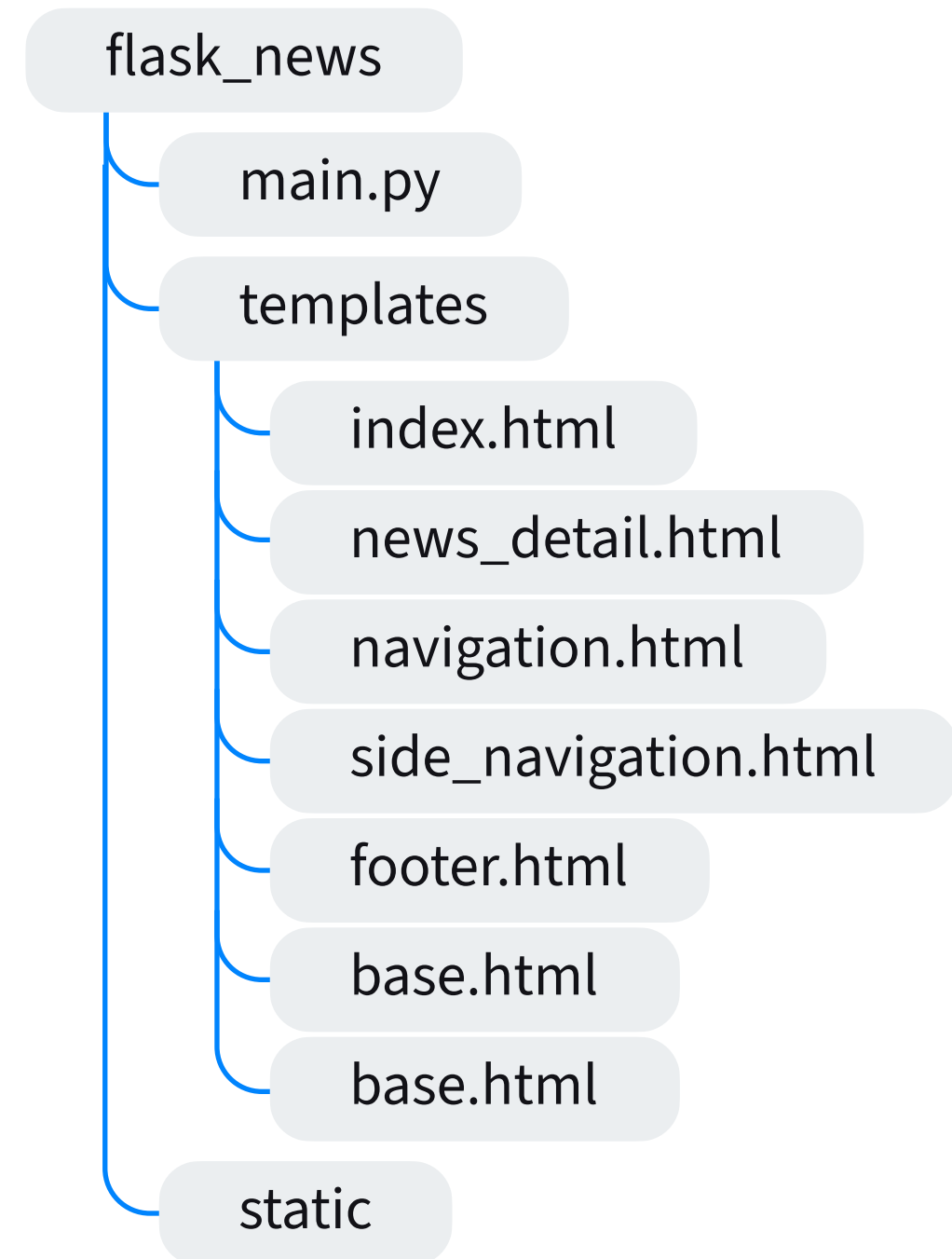
**Продолжаем создавать
новостной портал**

Повторение



Проверим структуру проекта flask_news и вспомним, на чем мы остановились.

Что нужно добавить в наш проект?



Выбор базы данных

В качестве базы данных для нашего проекта выберем базу данных SQLite.

SQLite — компактная кроссплатформенная встраиваемая (работает на том же компьютере и не требует отдельного сервера) база данных.

Python умеет работать с SQLite «из коробки» с помощью библиотеки `sqlite3`, а еще мы уже работали с ней в первом модуле, когда работали с PyQt.

ORM

ORM (Object-Relational Mapping) — это технология, позволяющая работать с реляционной базой данных с помощью концепций объектно-ориентированного языка программирования.



Object — объекты, которые представляют из себя классы.



Relational — реляционные базы данных.



Mapping — связь между объектами и базами данных.

При использовании ORM, в случае необходимости, можно перейти на другую базу данных — для этого потребуется только изменить настройки проекта.

SQLAlchemy

В Flask нет своей встроенной ORM, поэтому можно выбрать то, что подходит именно вашему проекту, например SQLAlchemy, PonyORM или что-то другое. SQLAlchemy была выпущена в 2006 году и с тех пор быстро завоевала сердца многих разработчиков на Python. Остановим свой выбор на ней.

SQLAlchemy — это библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM.

Подключим базу данных используя SQLAlchemy к нашему проекту.

Flask и SQLAlchemy

Чтобы использовать SQLAlchemy в Flask, можно воспользоваться готовой библиотекой flask-sqlalchemy. Эта библиотека позволяет упростить использование SQLAlchemy в Flask приложении, устанавливая связь между фреймворком и ORM.

Для установки flask-sqlalchemy в виртуальное окружение проекта выполните последовательно команды:

```
pip install SQLAlchemy==1.4.7
```

```
pip install Flask-SQLAlchemy==2.5.1
```

Подключение базы данных

...

```
from flask_sqlalchemy import SQLAlchemy
```

```
app = Flask(__name__)
```

...

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
```

```
db = SQLAlchemy(app)
```

...

Импортируем класс SQLAlchemy из flask_sqlalchemy и создаем экземпляр этого класса db, передав в качестве параметра экземпляр приложения app.

Чтобы подключить к SQLAlchemy базу данных SQLite, необходимо добавить настройку **SQLALCHEMY_DATABASE_URI** со значением **'sqlite:///db.sqlite3'**.

Подключение БД к flask_news

```
from flask import Flask, render_template, redirect, url_for
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
from flask_wtf import FlaskForm
```

```
from wtforms import StringField, TextAreaField, SubmitField
```

```
from wtforms.validators import DataRequired, Length
```

```
app = Flask(__name__)
```

```
app.config['SECRET_KEY'] = 'SECRET_KEY'
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
```

```
db = SQLAlchemy(app)
```

...

Модели

Модель — это класс, который необходимо создать и наследовать от класса `db.Model` (`db` — это экземпляр `SQLAlchemy`). Каждая модель это таблица базы данных.

Атрибуты класса (поля) — это столбцы таблицы в базе данных.

```
from datetime import datetime
```

```
...
```

```
db = SQLAlchemy(app)
```

```
class News(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    title = db.Column(db.String(255), unique=True, nullable=False)
```

```
    text = db.Column(db.Text, nullable=False)
```

```
    created_date = db.Column(db.DateTime, default=datetime.utcnow)
```

```
...
```

Стандартные типы

SQLAlchemy поддерживает множество типов, но при использовании SQLite можно использовать только некоторые. Рассмотрим самые распространенные:

Integer

Целое число

Boolean

Логический тип

String(size)

Строка с указанием наибольшей длины

Text

Строка любой длины

DateTime

Дата и время

Дополнительные параметры

Для полей также можно задать дополнительные параметры.

Например:

`primary_key`

Помечает этот столбец первичным ключом

`autoincrement`

Автоматическое увеличение первичного ключа

`unique`

Поле может быть только уникальным

`nullable`

Поле может быть пустым

`default`

Задаёт значение по умолчанию

Создание базы данных

...

```
app = Flask(__name__)  
app.config['SECRET_KEY'] = 'SECRET_KEY'  
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite3'
```

```
db = SQLAlchemy(app)
```

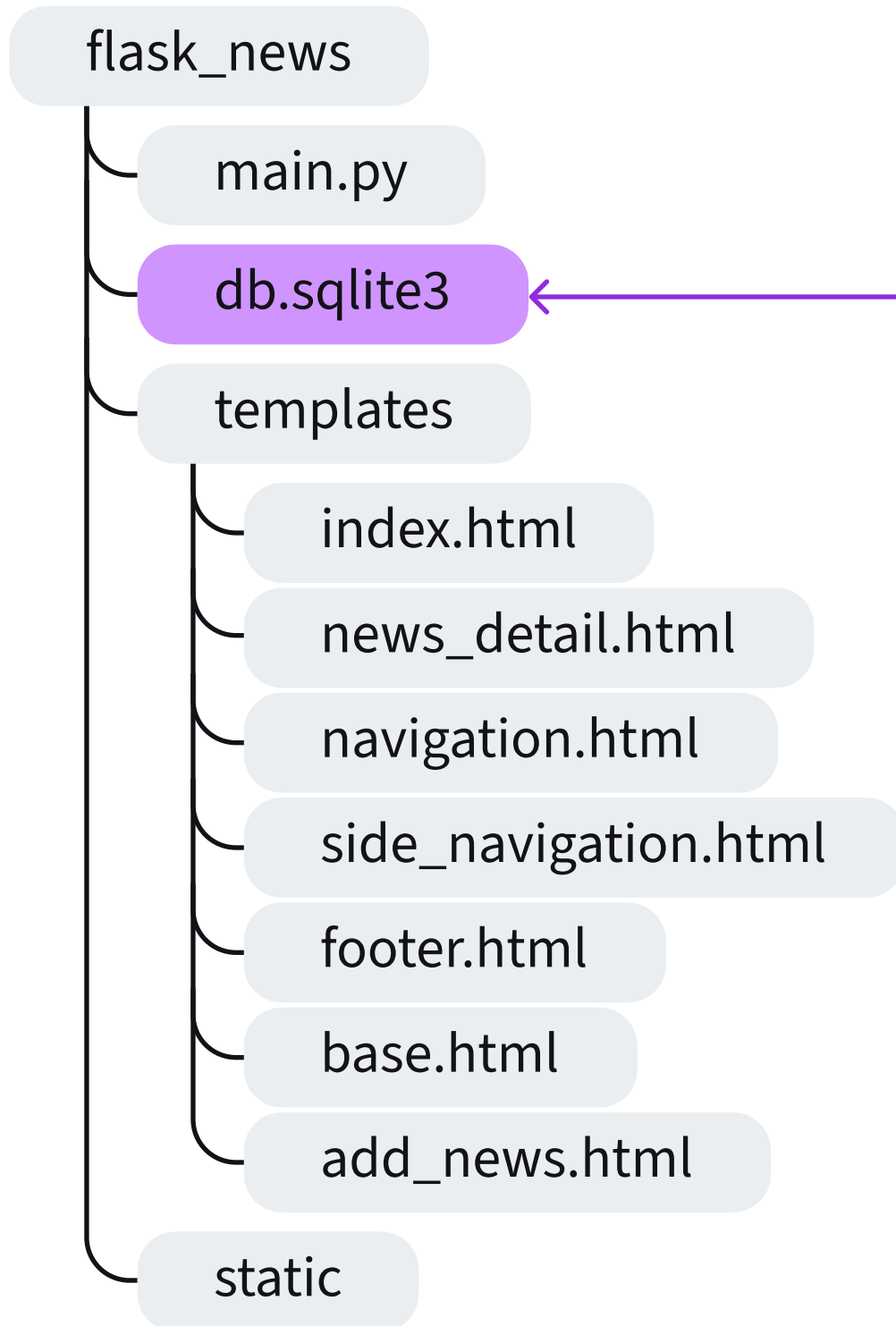
```
class News(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    title = db.Column(db.String(255), unique=True, nullable=False)  
    text = db.Column(db.Text, nullable=False)  
    created_date = db.Column(db.DateTime, default=datetime.utcnow)
```

```
db.create_all()
```

...

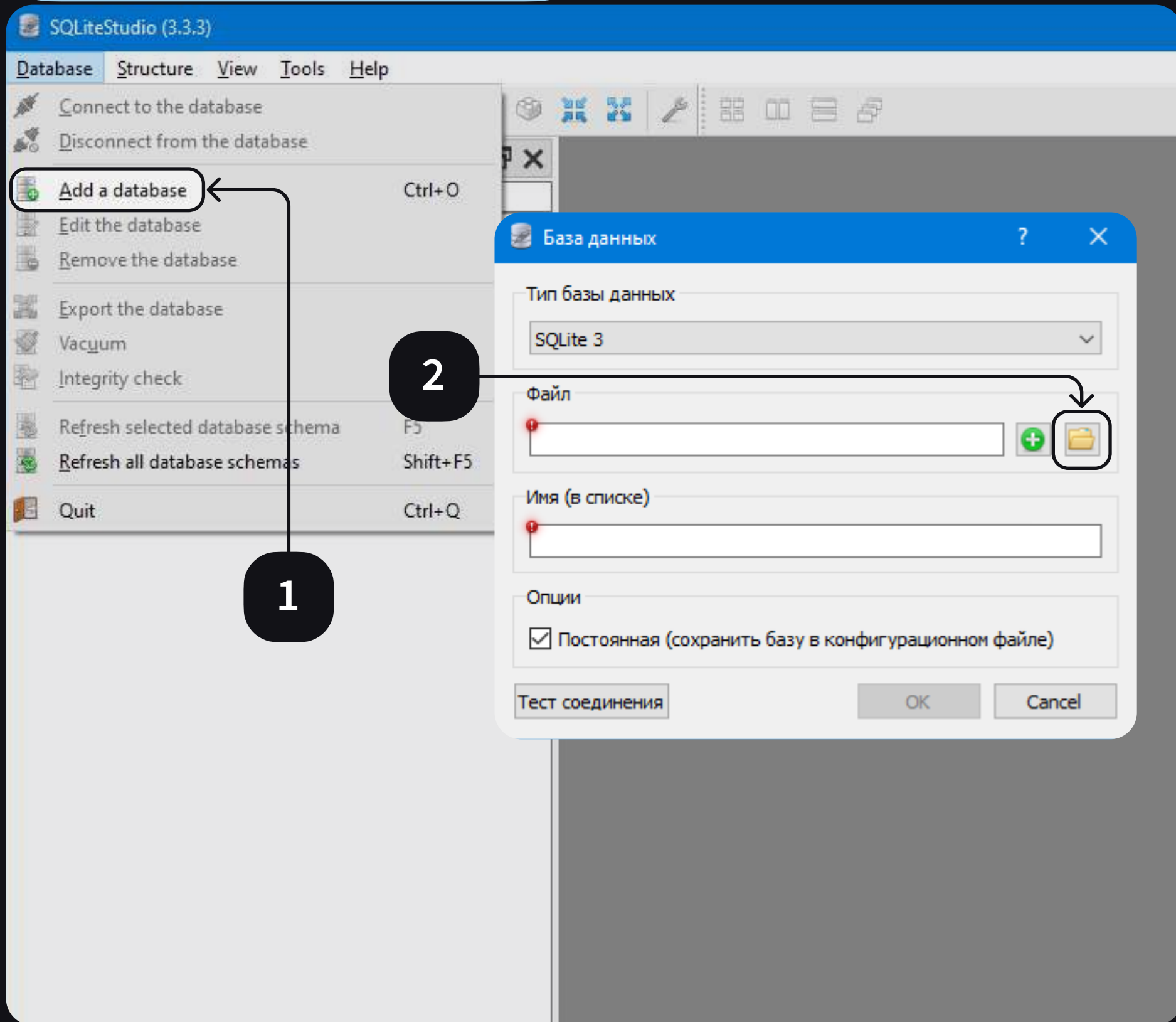
Для создания таблиц базы данных воспользуемся методом `create_all()`.

База данных



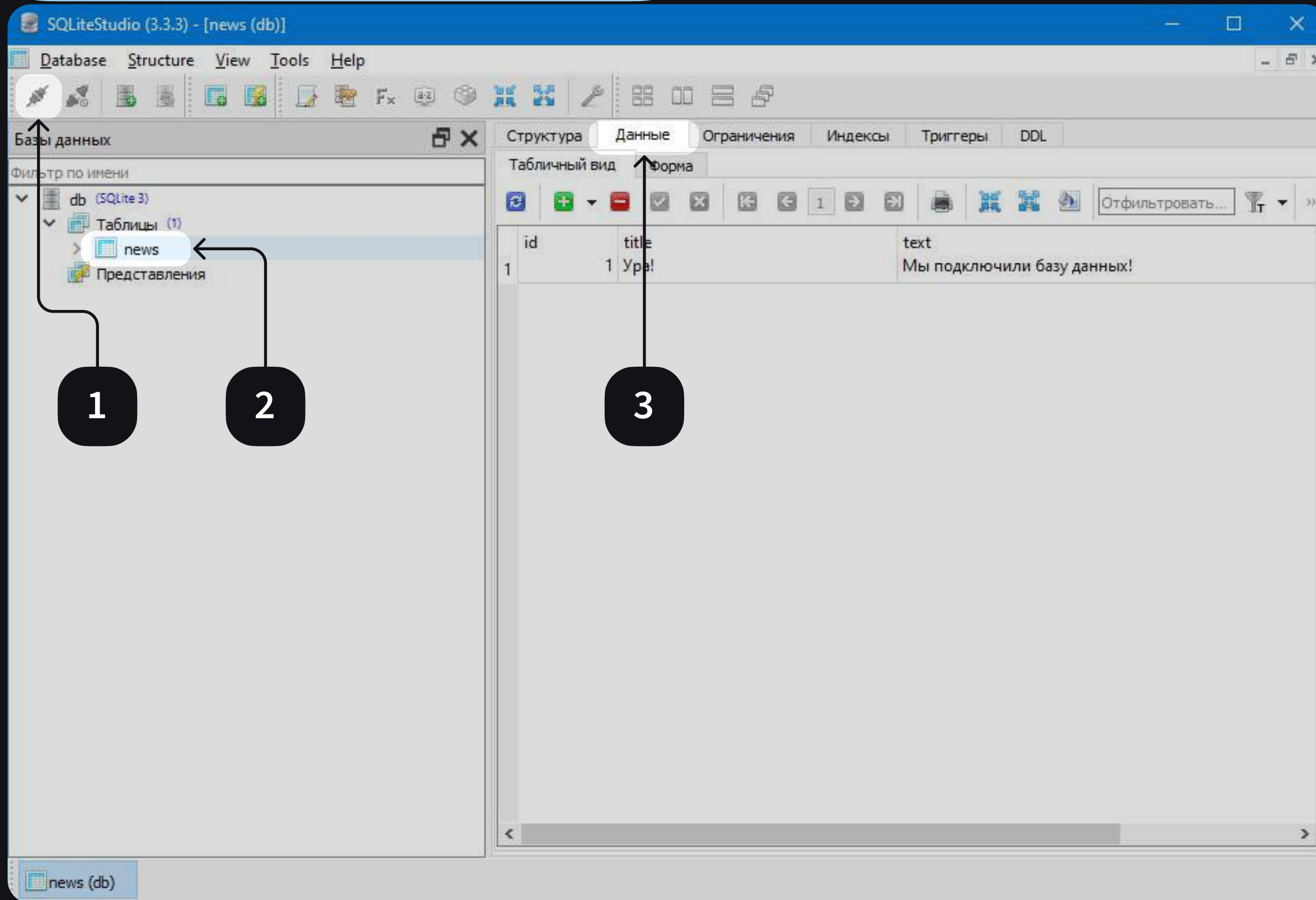
После запуска проекта
должен появиться файл
с базой данных `db.sqlite3`

SQLiteStudio



Для работы с базой данных SQLite можно воспользоваться программой SQLiteStudio. Скачайте и установите SQLiteStudio и добавьте созданную базу данных.

Работа с данными



Установите
соединение с базой
данных, откройте
созданную таблицу
и заполните ее
тестовыми данными.

Получение новостей

main.py:

...

```
@app.route('/')
```

```
def index():
```

```
    news_list = News.query.all()
```

```
    return render_template('index.html',  
                           news=news_list)
```

```
@app.route('/news_detail/<int:id>')
```

```
def news_detail(id):
```

```
    news_detail = News.query.get(id)
```

```
    return render_template('news_detail.html',  
                           news=news_detail)
```

У моделей есть атрибут `query`, обращаясь к которому можно получить объект запроса `Query`, соответствующий всем доступным записям модели.



Метод `all()` возвращает результаты, представленные этим `Query` объектом в виде списка.



Метод `get(id)` возвращает один объект БД (или `None`), значение первичного ключа которого равно `id`.

Проверим шаблон index.html

```
{% extends "base.html" %}
{% block title %}
    Главная страница
{% endblock title %}
{% block content %}
    <h1>Главная</h1>
    <hr>
    {% for one_news in news %}
    <div class="card mb-3">
        <div class="card-body">
            <h5 class="card-title">{{ one_news.title }}</h5>
            <p class="card-text">{{ one_news.text|truncate(100) }}</p>
            <a href="{{ url_for('news_detail', id=one_news.id) }}" class="btn btn-primary">Читать далее</a>
        </div>
    </div>
    {% else %}
    <p class="card-text">Новостей пока нет!</p>
    {% endfor %}
{% endblock content %}
```

Значительных изменений в шаблоне нет, самое главное изменение — это формирование ссылки, теперь параметр `id` — это первичный ключ записи в таблице.

Этот блок будет выводиться если список `news` будет пуст.

Проверим шаблон news_detail.html

```
{% extends "base.html" %}
```

```
{% block title %}
```

```
  {{ news.title }}
```

```
{% endblock title %}
```

```
{% block content %}
```

```
  <h1>{{ news.title }}</h1>
```

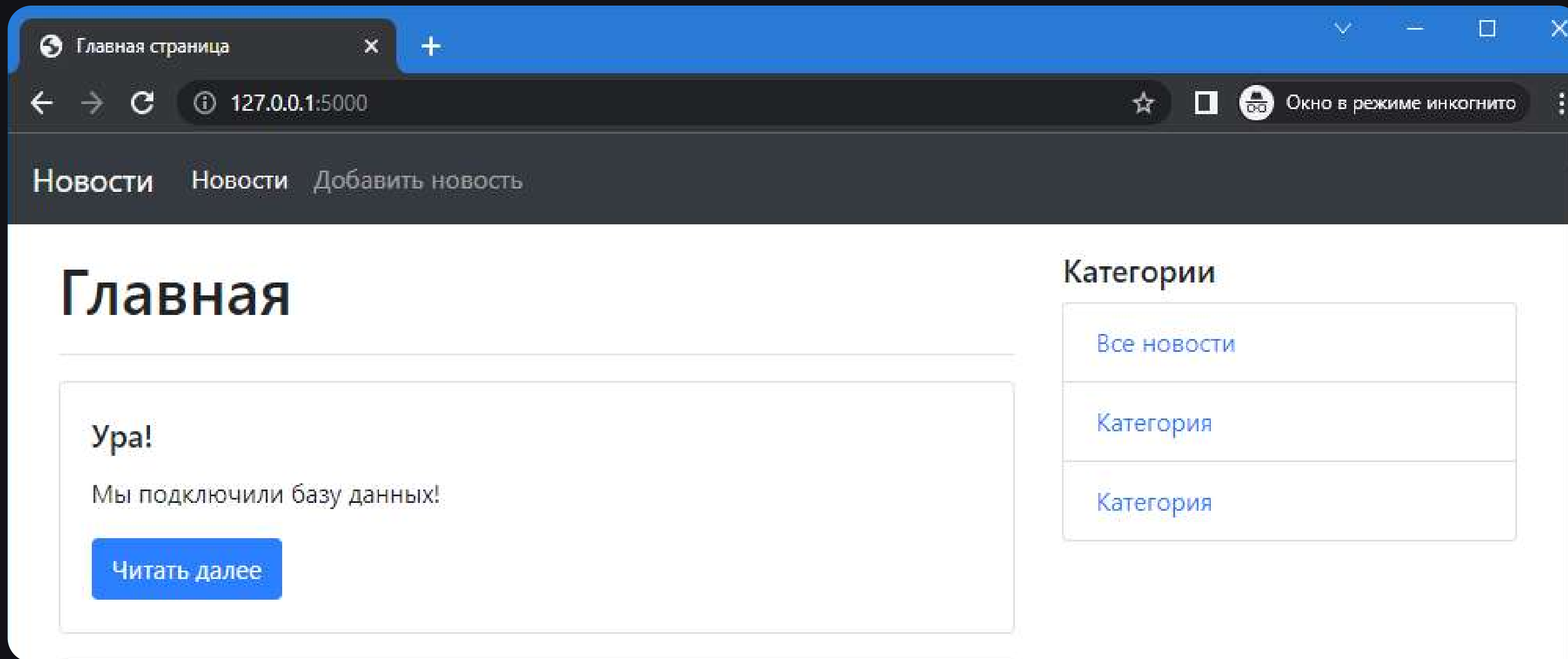
```
  <hr>
```

```
  <p>{{ news.text }}</p>
```

```
{% endblock content %}
```

Теперь в шаблон news_detail.html передается объект News, а не конкретные переменные. Выведем его атрибуты.

Результат



Запустите приложение, проверьте, что в приложении отображаются новости из базы данных и ссылки работают корректно.

Добавление новостей

main.py:

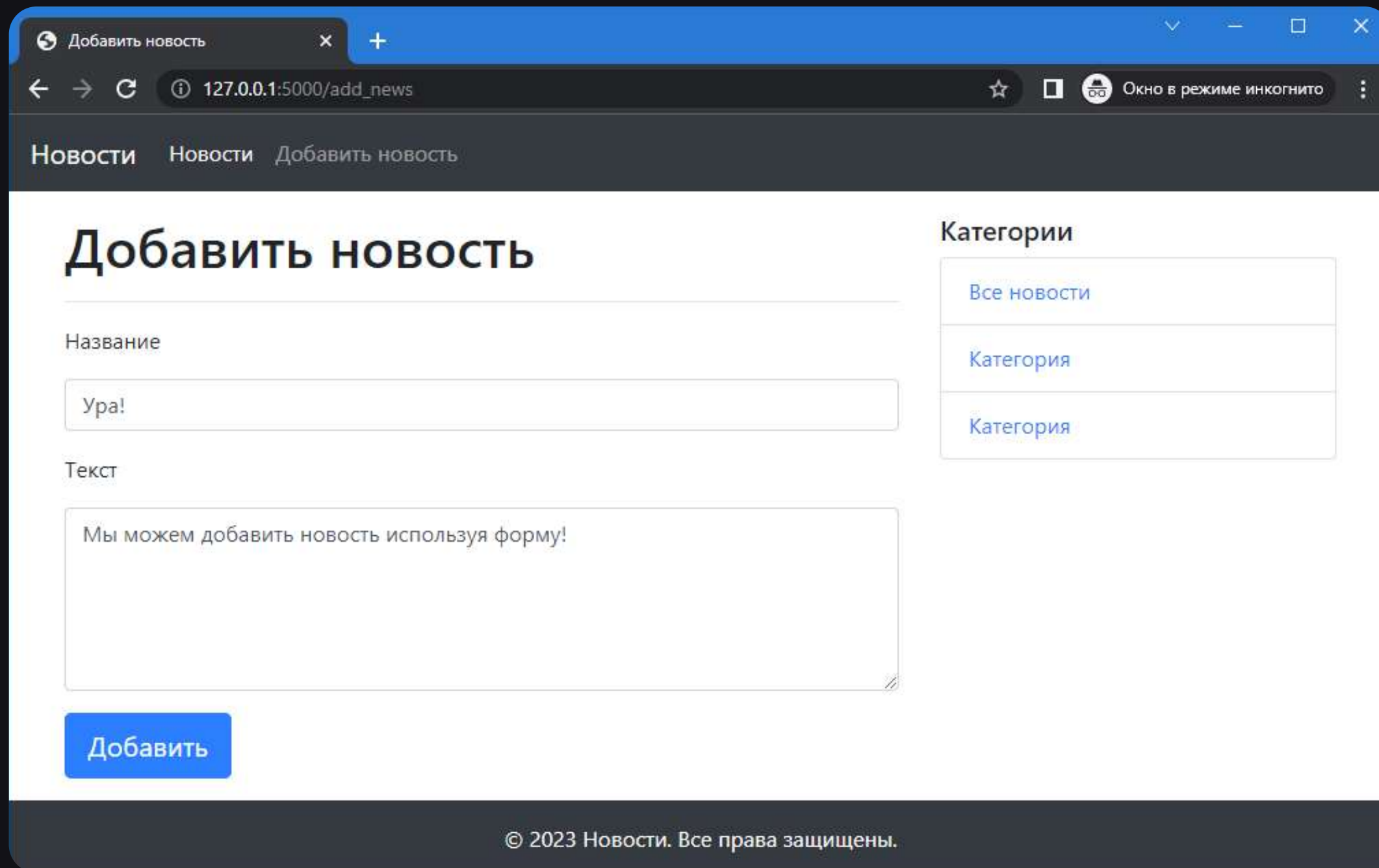
...

```
@app.route('/add_news', methods=['GET', 'POST'])
def add_news():
    form = NewsForm()
    if form.validate_on_submit():
        news = News()
        news.title = form.title.data
        news.text = form.text.data
        db.session.add(news)
        db.session.commit()
        return redirect(url_for('news_detail', id=news.id))
    return render_template('add_news.html',
                           form=form)
```

Для добавления новой записи создадим экземпляр класса **News** и укажем значения обязательных атрибутов — это **title** и **text**. Аtribуты **id** и **created_date** будут сформированы автоматически.

Для добавления объекта выполняем команды **db.session.add(news)** и **db.session.commit()** — для подтверждения сохранения изменений в базе данных.

Результат



Добавить новость

127.0.0.1:5000/add_news

Новости Новости Добавить новость

Добавить новость

Название

Ура!

Текст

Мы можем добавить новость используя форму!

Добавить

Категории

- Все новости
- Категория
- Категория

© 2023 Новости. Все права защищены.

Запустите приложение и попробуйте добавить новость, используя форму. После нажатия кнопки «Добавить» новость попадает в базу данных, а пользователь должен быть переадресован на страницу с новостью.

Итоги



Используя технологию **ORM**, можно использовать различные базы данных, не изменяя существенно код проекта.



ORM (Object-Relational Mapping) — это технология, позволяющая работать с реляционной базой данных с помощью концепций объектно-ориентированного языка программирования.



SQLAlchemy — это библиотека на языке Python для работы с реляционными СУБД с применением технологии ORM.



Для использования SQLAlchemy в Flask можно воспользоваться библиотекой flask-sqlalchemy.



Для создания таблиц в базе данных необходимо описать модели — Python классы.