Хекслет

Что такое Flask и как на нем быстро написать простой сайт

Python ~16 минут

Рассказываем про один из самых популярных и лаконичных микрофреймворков для Python — Flask. Как написать простое приложение, подключить к нему Bootstrap и базу данных, и есть ли вообще у Flask минусы.

Flask — это микрофреймворк для создания веб-приложений на Python. В нем из коробки доступен только минимальный инструментарий, но при этом он поддерживает расширения так, как будто они реализованы в самом Flask. Расширения для микрофреймворка позволяют коммуницировать с базами данных, проверять формы, контролировать загрузку на сервер, работать с аутентификацией и многое другое.

Первая публичная версия Flask вышла 16 апреля 2010 года. Автор проекта — Армин Ронахер, который возглавлял команду энтузиастов в

Python-разработке Pooccо. Flask основан на быстром и расширяемом механизме шаблонов Jinja и наборе инструментов Werkzeug. Кроме того, Flask использует одну из самых передовых служебных библиотек WSGI (Web Server Gateway Interface — стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером).

При этом WSGI тоже разработал Армин Ронахер. По его словам, идея Flask изначально была первоапрельской шуткой, которая стала популярной и превратилась в серьезное приложение.

Плюсы и минусы Flask

Практически все плюсы и минусы Flask появились именно из-за того, что он является микрофреймворком.

Среди достоинств:

- **Простота.** Flask легко установить и настроить.
- **Гибкость.** Микрофреймворк позволяет разработчикам самостоятельно выбирать технологии и инструменты, которые они хотят применять в своих проектах
- **Расширяемость.** Flask позволяет расширять функциональность с помощью плагинов и модулей, которые можно легко интегрировать в проект.
- **Активное сообщество.** Flask является одним из самых используемых фреймворков для Python, поэтому имеет большое комьюнити разработчиков.

При этом у Flask есть и свои недостатки:

- **Отсутствие готовых решений.** Разработчики изначально могут использовать во Flask только минимальный набор функциональности. Если же программисту нужны более широкие возможности, такие как аутентификация пользователя, ему придется добавить дополнительные библиотеки или реализовать это самостоятельно.
- **Нет встроенной многопоточности.** Flask был разработан как однопоточный фреймворк. И чтобы управлять многопоточными веб-приложениями, придется установить дополнительные библиотеки.
- **Ограниченные возможности для масштабирования.** Если проект начинает расти и усложняться, то могут появиться сложности в поддержке нужной функциональности.

То есть Flask можно удобно использовать в небольших проектах — он идеален для макетирования идей и быстрого прототипирования. При этом его редко используют в крупных проектах, и он плохо подходит для асинхронного программирования.

Как создать проект на Flask

Для начала работы с микрофреймворком нужно скачать последнюю версию Flask:

```
pip install Flask
```

Для примера мы напишем на Flask тестовое веб-приложение с минимальным функционалом. Как работают приложения такого типа:

- Пользователь вводит в браузере url, например — hexlet.io. В нашем тестовом приложении пользователь не будет вводить url, потому что мы будем работать с локальным сервером по адресу `http://127.0.0.1:5000`.
- Браузер получает у DNS IP-адрес нужного нам сервера. DNS — это Domain Name System, распределенная системе серверов. Она работает как общая «контактная книга» в интернете.
- Браузер отправляет запрос по этому адресу и получает ответ. Как правило — в виде HTML-страницы.

- Браузер отображает содержимое страницы.

Итак, создадим файл `hello.py` и запишем в него следующий код:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index() -> str:
    return '<p>Hello Flask!</p>'

if __name__ == '__main__':
    app.run(debug=True)
```

Давайте подробно разберем, что делает код, который мы написали.

Первой строкой мы импортируем класс `Flask`. После этого мы создаем объект этого класса, передав первым аргументом имя модуля, — это и будет наше приложение для общения с веб-сервером. `__name__` — это удобный способ передать именно то приложение, из которого запущен Flask.

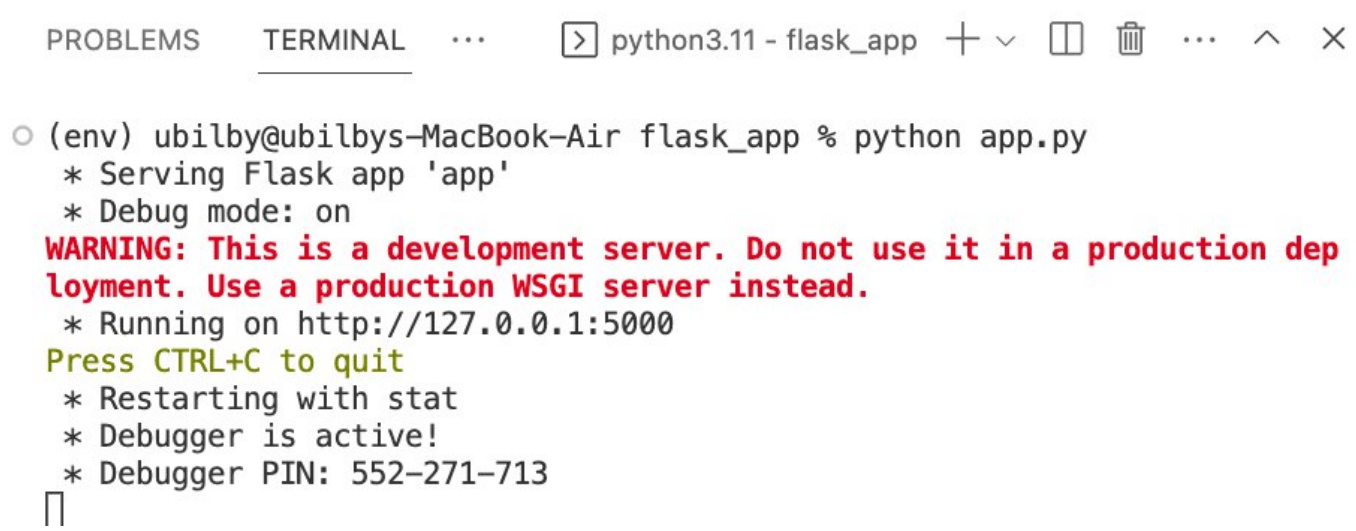
Декоратор `route()` сообщает Flask, при обращении к какому URL-адресу запустится декорируемая разработчиком функция — в нашем примере это `index`. Последней строчкой мы открываем локальный веб-сервер с параметром `debug=True` — это позволит следить за всеми ошибками в логе программы.

> Читайте также:
>
> Программирование на Python: особенности обучения, перспективы, ситуация на рынке труда

Запускаем веб-приложение через терминал:

```
python hello.py
```

Если мы все сделали правильно, то в терминале появятся эти сообщения:



В основном тут отображается служебная информация. Единственное, что нас интересует — сообщение, что наш локальный сервер запущен по адресу http://127.0.0.1:5000/. В нем красными буквами указывается, что локальный сервер не подходит для продакшена. Но, так как мы реализовали тестовое приложение, то не будем деплоить его на реальный сервер.

Вернемся к коду. С помощью переменной части маршрута Flask может передавать в функцию аргументы.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    greet_with_link = """<h1>Привет, Мир!</h1>
    <p><a href="user/Аникин/Георгий">Нажми на меня</a></p>"""
    return greet_with_link

@app.route('/user/<surname>/<name>')
def get_user(name, surname):
    personal_instruc = f"""<h1>Привет, {surname} {name}!</h1>
    <p>Измени имя пользователя в адресной строке и перезагрузи
    return personal_instruc
```

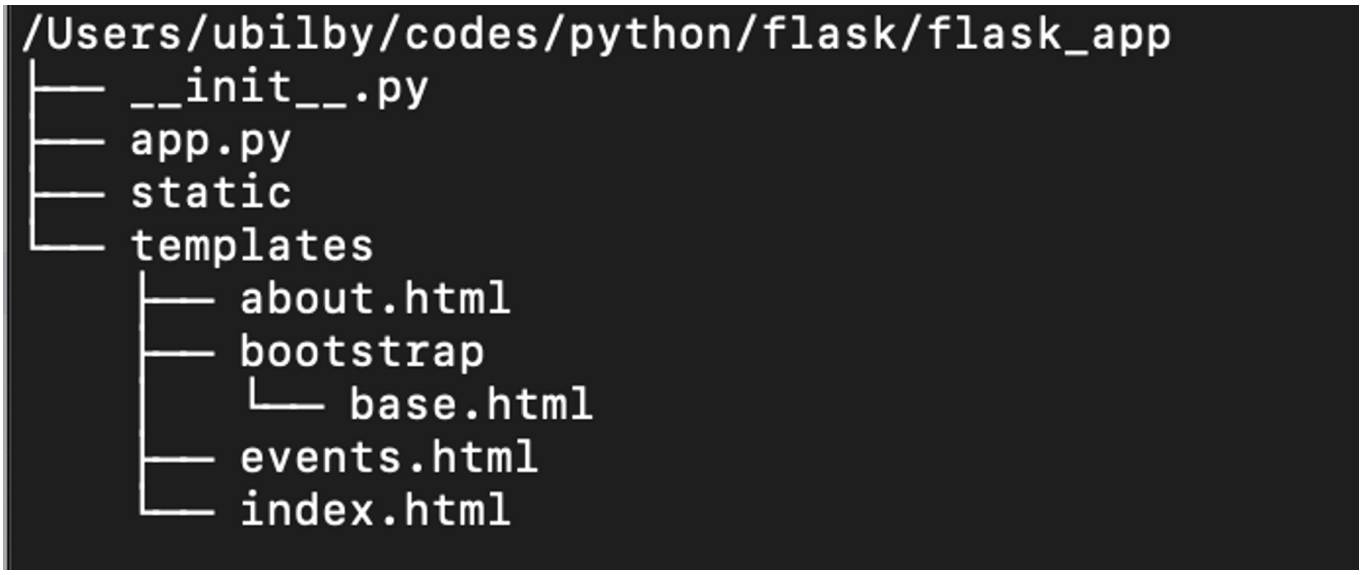
```
if __name__ == '__main__':
    app.run(debug=True)
```

В нашем примере значения просто появятся в браузере как часть строки. На стартовой странице нашего сайта будет запускаться функция `index()`. В ней пользователю, помимо приветствия, будет предлагаться нажать на ссылку, при клике на которую он перейдет на `user/Аникин/Георгий`. Этот URL-маршрут будет обрабатываться уже функцией `get_user`.

Функция `get_user` декорируется `@app.route('/<surname>/<name>')`, а в адресной строке у нас `/user/Аникин/Георгий`. То есть наша функция получает аргументы из URL-адреса, эти значения лежат между косых скобок. По умолчанию тип таких значений `string` принимает любой текст без косой черты. Но переменные маршрутов могут быть и иных типов: `int`, `float`, `path` и других. Типы указываются в формате `<тип:имя переменной>`.

Структура приложения на Flask

Создадим подкаталог `flask_app` с такой структурой файлов и папок:



Чтобы написать приложение сложнее одной строки, в директории проекта должны находиться папки `static` и `templates`. Директория `static` содержит ресурсы, которые используются шаблонами. В том числе включая

файлы CSS, JavaScript и картинки. Папка `templates` содержит только шаблоны с расширением `*.html`.

Заполним наши файлы кодом. Сначала — наш основной файл проекта `app.py`:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")


@app.route("/about")
def get_page_about():
    return render_template("about.html")

if __name__ == "__main__":
    app.run(debug=True)
```

После этого — `index.html`:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Main page</title>
    </head>
    <body>
        <h1>Главная страница</h1>
    </body>
</html>
```

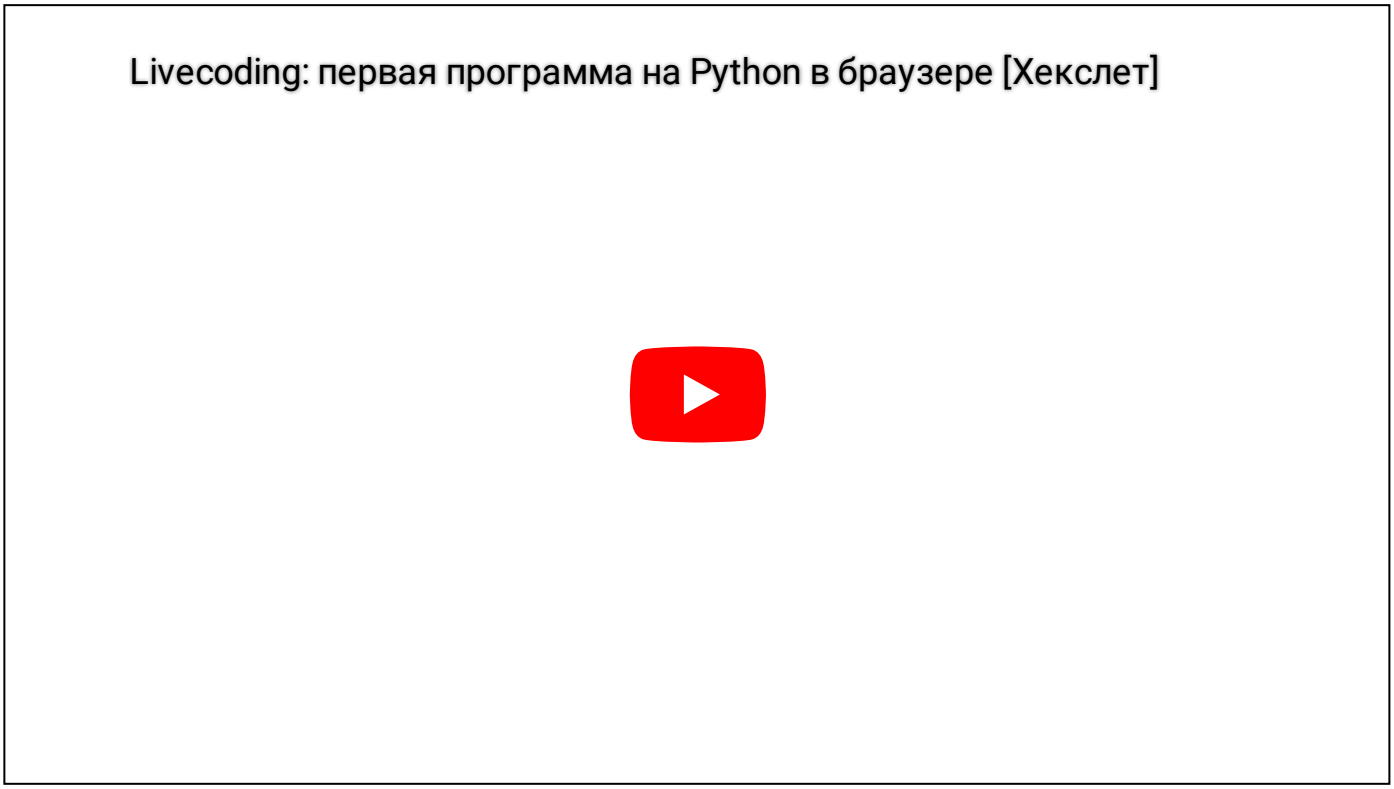
И файл `about.html`:

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>About</title>
    </head>
    <body>
        <h1>О приложении</h1>
    </body>
</html>
```

Для отображения HTML-шаблонов мы используем функцию `render_template()`. В нашем коде она принимает только имя шаблона и возвращает строку с результатом рендеринга шаблона.

Однако `render_template()` может принимать неограниченное количество именованных аргументов, которые можно использовать в этом шаблоне. Это позволит решить проблему нашего тестового проекта — сейчас у нас две функции, две страницы, и очень много дублированного кода.



Напишем базовый шаблон `base.html` и пару его наследников. При этом блоки `{% block smth %} … {% endblock %}` — это части базового шаблона, которые можно заменить в наследнике. Переменные передаются по именам в конструкции `{{ variable }}`.

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}Приложение Flask{% endblock %}</t
</head>
<body>
    <h1>{{h1}}</h1>
</body>
</html>
```

После появления файла с базовым HTML-шаблоном можем поправить наши остальные HTML-файлы:

```
about.html:
{% extends 'base.html' %}
{% block title %}About{% endblock %}

index.html:
{% extends 'base.html' %}
{% block title %}Main page{% endblock %}
```

Кроме того, нужно поправить и основной файл Flask-проекта app.py:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html", h1 = "Главная страни

@app.route("/about")
def get_page_about():
    return render_template("about.html", h1 = "О приложении")
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

Подключаем Bootstrap

Bootstrap — это открытый и бесплатный набор инструментов для создания сайтов и веб-приложений.

В нашем проекте в папке templates у нас есть подкаталог bootstrap, а в нем файл base.html — это немного модифицированная заготовка сайта-документации Bootstrap-Flask:

```
<!doctype html>
<html lang="en">
    <head>
        {% block head %}
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, in

        {% block styles %}
            <!-- Bootstrap CSS -->
            {{ bootstrap.load_css() }}
        {% endblock %}

        <title>{% block title %}Приложение Flask{% endblock %
        {% endblock %}
    </head>
    <body>
        <!-- Your page content -->
        {% block content %}
            <div class="jumbotron text-center">
            <h1>{{h1}}</h1>
            </div>
        {% endblock %}

        {% block scripts %}
```

```
            <!-- Optional JavaScript -->
            {{ bootstrap.load_js() }}
        {% endblock %}
    </body>
</html>
```

В файлах `index.html` и `about.html` заменим строку наследования на:

```
{% extends 'bootstrap/base.html' %}
```

Второй путь подключения Bootstrap к проекту на Flask — через CDN. Подробнее об этом можно почитать в документации фреймворка.

> ### Читайте также:
>
> Как создатель Python Гвидо ван Россум устроился в Microsoft и теперь работает над развитием CPython

После подключения Bootstrap нужно будет немного поправить основной файл нашего проекта `app.py`:

```
from flask_bootstrap import Bootstrap4
from flask import Flask, render_template

app = Flask(__name__)
bootstrap = Bootstrap4(app)

@app.route("/")
def index():
    return render_template("index.html", h1 = "Главная страни

@app.route("/about")
def get_page_about():
    return render_template("about.html", h1 = "О приложении")
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

Последним элементом нашего веб-приложения будет форма отправки. Для этого нужно немного модифицировать `index.html`:

```
{% extends 'bootstrap/base.html' %}
{% block title %}Main page{% endblock %}
{% block content %}
   {{super()}}
   <div class="container text-center">
       <form class="d-inline-block" style="max-width: 33%;">
           <div class="form-group">
               <label for="eventDate">Дата</label>
               <input type="date" name="eventDate" class="forr
           </div>
           <div class="form-group">
               <label for="eventName">Событие</label>
               <input type="text" name="eventName" class="forr
           </div>
           <div class="form-group">
               <label for="eventDuration">Продолжительность</l
               <input type="number" name="eventDuration" class
           </div>
           <button type="submit" class="btn btn-primary">Запи
       </form>
   </div>
{% endblock %}
```

Вообще, Bootstrap может добавить огромное количество элементов в приложение буквально в несколько кликов. Мы ограничились четырьмя — три поля и одна кнопка. Ключевой элемент здесь — это `{{ super() }}`.

Подключаем базу данных

Итак, у нас есть форма отправки, но она пока ничего не делает с данными. Для нас было бы неплохо хранить, обрабатывать и в будущем легко извлекать данные этих форм. Обычно такие задачи решают с помощью реляционных баз данных (далее БД).

Есть большое количество способов работы с SQL-запросами во Flask. Мы можем использовать, например, sqlite3 и чистый SQL, а можем — библиотеку sqlite3 для Python. Кроме того, можно обернуть чистые SQL-запросы в код, либо использовать Psycopg2 для работы с PostgresSQL в Python (мы рекомендуем делать именно так и вот почему). Для примера в этом тексте мы используем библиотеку Flask SQLAlchemy (расширение для Flask), которая предлагает технологию ORM для взаимодействия с БД.

Подключаем базу данных к нашему проекту через файл `app.py`:

```
from datetime import datetime


from flask import Flask, redirect, render_template, request
from flask_bootstrap import Bootstrap4
from flask_sqlalchemy import SQLAlchemy


app = Flask(__name__)
bootstrap = Bootstrap4(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///events.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)


class Event(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    date = db.Column(db.Date, nullable=False)
    name = db.Column(db.String(255), nullable=False)
    duration = db.Column(db.Integer, nullable=False)

    def __str__(self):
        return (
```

```
            f"Название: {self.name}\n"
            f"Дата: {self.date}\n"
            f"Продолжительность {self.duration}ч"
        )


@app.route('/', methods=['POST'])
def add_event():
    date = datetime.strptime(request.form['eventDate'], '%Y-%m-
    name = request.form['eventName']
    duration = int(request.form['eventDuration'])
    print(date, name, duration, sep='\n')
    event = Event(date=date, name=name, duration=duration)
    db.session.add(event)
    db.session.commit()
    return redirect('/')


@app.route("/")
def index():
    return render_template("index.html", h1 = "Главная страница


@app.route("/about")
def get_page_about():
    return render_template("about.html", h1 = "О приложении")


if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

В нашей БД появился класс `Event` с атрибутами, который наследуется от `db.Model`. Это позволяет с помощью SQLAlchemy создать таблицу `event`, а поля нашего класса сделать колонками этой таблицы. Кроме того, мы определили магический метод `__str__` для строкового отображения экземпляров класса — это пригодится для отображения в HTML.

Для создания таблицы в блок `if __name__ == '__main__'` мы добавили команду `db.create_all()`, а для обработки отправленной формы – метод `add_event`. Он работает с методом `POST`, который указывает Flask, что данные будут отправлены на сервер.

В методе `POST` мы считываем данные отправленной формы и создаем для каждой строки временную переменную. После этого мы создаем объект `event` класса `Event`, передаем наши временные переменные как именованные аргументы, добавляем `event` в БД и фиксируем изменения.

Нам осталось лишь немного поправить форму: в файле `index.html` в открывающем теге `<form>` добавим атрибуты `action="{{ url_for('add_event') }}" method="POST"`. Теперь форма отправки по нажатию на кнопку «Записать» будет отправлять данные в базу данных.

Добавим страницу отображения наших записей в новый файл `Events.html`:

```
{% extends 'bootstrap/base.html' %}
{% block title %}Events{% endblock %}
{% block content %}
   {{super()}}
   <div class="container text-center">
       <a href="{{ url_for('index') }}"><h2>Добавить событие</
   </div>
   <div class="mt-4">
       <ul class="list-group">
           {% for event in events %}
               <li class="list-group-item">
                   <p>{{ event }}</p>
               </li>
           {% endfor %}
       </ul>
   </div>
{% endblock %}
```

В файл `app.py` добавим `view`:

```
@app.route("/events")
def view_events():
    events = Event.query.order_by(Event.date).all()
    return render_template("events.html", h1 = "События", event
```

А в основном контейнере `index.html` добавим ссылку на эту страницу:

```
<a href="{{ url_for('view_events') }}"><h2>Посмотреть события<
```

Наш тестовый проект на Flask готов! Его можно запустить на локальном сервере через команду `python app.py` (в некоторых случаях надо будет написать название директории перед названием файла `app.py`).

Что еще почитать про Flask

- Большой курс по Flask на Хекслете
- Документация Flask
- Цикл статей на Real Python
- Проектирование RESTful API с помощью Python и Flask

Георгий Аникин
03 апреля 2023

6

Рекомендуемые программы

PHP-разработчик

Разработка веб-приложений на Laravel

10 месяцев · с нуля Старт 5 октября

профессия · от 6 183 ₽ в месяц

Инженер по тестированию

Ручное тестирование веб-приложений

4 месяца · с нуля Старт 5 октября

профессия · от 6 300 ₽ в месяц

Node.js-разработчик

Разработка бэкенд-компонентов для веб-приложений

10 месяцев · с нуля Старт 5 октября

профессия · от 10 080 ₽ в месяц

Fullstack-разработчик

Разработка фронтенд- и бэкенд-компонентов для веб-приложений

16 месяцев · с нуля Старт 5 октября

профессия · от 5 840 ₽ в месяц

Разработчик на Ruby on Rails

Создание веб-приложений со скоростью света

5 месяцев · с опытом Старт 5 октября

профессия

Верстальщик

Верстка с использованием последних стандартов CSS

5 месяцев · с нуля

Старт в любое время

профессия

новый

Инженер по автоматизированному тестированию на JavaScript

Автоматизированное тестирование веб-приложений на JavaScript

в разработке

8 месяцев · с опытом

дата определяется

Курсы «Frontend-разработка»

Курсы «Создание сайтов»

Курсы «Тестирование»

Курсы «Аналитика данных»

Интенсивные курсы

Курсы DevOps

Курсы «Веб-разработка»

Курсы «Математика для программистов»

Курсы JavaScript

Курсы Python

Курсы Java

Курсы PHP

Курсы Ruby

Курсы Go

Курсы HTML

Курсы SQL

Курсы Git

Корпоративное обучение

Все категории

Документы

Условия использования

Соглашение об обработке ПД

Оферта

Акции

Помощь

Справка

Вопросы и ответы

Сообщество

Дополнительно

О Хекслете

О нас

Карьера в Хекслете

Хекслет Колледж

Сведения об образовательной организации

Лицензия № Л035-01216-73/00668844

ООО «Хекслет Рус»
432071, г. Ульяновск,
пр-т Нариманова, дом 1Г, оф. 23
ОГРН 1217300010476

Язык