Разработка Web приложений с Python и Flask

Дмитрий Купцов
2021

Оглавление

Введение

Веб приложения сегодня являются одним из самых распространённых способов создания сервисов, которые используются миллионами пользователей. Веб приложения легко обновлять, переносить, и распространять - они не требуют установки специальных средств на рабочей машине пользователей, кроме веб браузера.

В этой книге мы рассмотрим как создавать большие и структурированные веб приложения используя Python и библиотеку Flask. Мы рассмотрим такие вещи как безопасность веб приложений, базы данных, веб формы и структурирование больших веб приложений.

Данная книга предназначена для начинающих веб программистов, а также для всех тех, кто желает познакомиться с веб программированием. Мы предполагаем, что читатель уже ознакомился с нашей книгой об основах Python и алгоритмах. Данную книгу можно скачать бесплатно здесь [1].

ГЛАВА 1

Компоненты

Современное веб приложение состоит из множества компонентов: системы управления базами данных, или СУБД (как NoSQL, так и SQL), веб форм, контроллеров и моделей, клиентской части приложения, написанной на JavaScript, подсистемы безопасности и многочисленных пользовательских библиотек. В данной главе мы рассмотрим основные компоненты, которые мы будем использовать при построение нашего приложения во второй части нашей книги.

1.1 Среда окружения

Часто на рабочей машине может быть запущено несколько проектов одновременно. Каждый проект должен иметь свои зависимости и установленные библиотеки. Для того, чтобы не было путаницы в библиотеках на рабочей машине, для разработки, устанавливают виртуальное окружение, а также нужные библиотеки.

На дистрибутиве Linux Ubuntu 20.04 (мы предполагаем наличие Python 3.8.5) среда может быть установлена следующим образом.

Для начала необходимо установить пакет, который позволит работать с виртуальным окружением:

```
sudo apt-get install python3-venv
```

Далее создадим окружение:

```
$ python3 -m venv book_ml
```

После нужно активировать виртуальную среду разработки:

```
$ source book_ml/bin/activate
```

Далее можно устанавливать необходимые библиотеки и они не будут пересекаться с другими проектами:

```
$ pip3 install pycryptodome
```

1.2 Безопасность

Безопасность в веб приложениях является одним из основных вопросов. Популярность веб приложений и их повсеместность накладывает определённые требования к безопасности. В данной главе мы познакомим читателя с такими атаками как Cross Site Request Forgery (CSRF), неаутентифицированный и неавторизованный доступ к ресурсам, неверный ввод данных, а также рассмотрим методы защиты от таких атак. И конечно, мы затронем тему шифрования канала от пользователя до веб сервера с помощью Secure Socket Layer (SSL).

1.2.1 CSRF

Очень часто можно послать запрос на сайт замаскированный как сторонний и произвести какую либо транзакцию так, чтобы пользователь этого не заметил. Например, пусть пользователь авторизовался на вашем сайте, получил куки и не вышел из системы. Позже злоумышленник может прислать пользователю на почту картинку с изображением, например, кошечки и попросил перейти по ссылке. Ссылка же на самом деле ведёт к вашему сайту и автоматически пошлёт все куки файлы с запросом.

Это опасно тем, что если система не защищена от CSRF атак, то запрос выполнится и атакующий сможет изменить содержимое базы данных. Для того чтобы этот тип атаки не смог быть реализован, необходимо в форме HTML вставлять некий токен безопасности. Вместе с тем, тот же токен необходимо хранить в зашифрованном виде в куке файле. Если на сервер придет куки файл, и токен в нем будет отличаться от токена, полученного в форме, то запрос стоит отвергнуть, так как он небезопасный.

Приведём пример того, как можно использовать CSRF защиту в Flask приложении.

Для начала установим библиотеку для работы с формами и сам Flask:

```
$ pip3 install flask
$ pip3 install flask_wtf
```

Далее нужно сконфигурировать наше Flask приложение для работы с CSRF защитой. Для этого вставим следующие строки в наше приложение. Сначала объявим форму:

```
from flask_wtf import Form

class TestForm(Form):
  pass
```

А затем объявим контроллер, в котором данная форма будет отрисовываться:

```
from flask import Flask, render_template, redirect, url_for, jsonify
from flask_wtf.csrf import CSRFProtect
from forms import TestForm

app = Flask()

csrf = CSRFProtect(app)

app.config["SECRET_KEY"] = "gliNuryoc6";

@app.route("/")
def index():
    form = TestForm()
    return render_template("index.html", form=form)

app.run(port=8080, debug=True);
```

В веб файле index.html прописываем скрытое поле, которое будет содержать наш CSRF токен:

```
<form method="post">
    {{ form.csrf_token }}
</form>
```

Если вы используете jQuery AJAX запросы то нужно с запросом посылать и токен:

```
<script type="text/javascript">
    var csrf_token = "{{ csrf_token() }}";
```

```
    $.ajaxSetup({
        beforeSend: function(xhr, settings) {
            if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !
                                                    this.crossDomain) {
                xhr.setRequestHeader("X-CSRFToken", csrf_token);
            }
        }
    });
</script>
```

Если проверка верности токена будет нейдачной, Flask выбросит ошибку CSRFError. По умолчанию, Flask вернет HTTP с кодом 400 и объяснением причины ошибки. Если вы хотите отправить своё сообщение об ошибке то стоит зарегистрировать соответствующий обработчик ошибки:

```
from flask_wtf.csrf import CSRFError

@app.errorhandler(CSRFError)
def handle_csrf_error(e):
    return render_template('csrf_error.html', reason=e.description),
                                            400
```

Стоит заметить CSRF защита требует секретный ключ для подписи токена. По умолчанию Flask будет использовать SECRET_KEY переменную для этих целей. Если же вы захотите использовать отдельный ключ для этих целей, то можно установить переменную WTF_CSRF_SECRET_KEY. Требуется использовать достаточно большой ключ для шифрования - ключ, который будет содержать достаточно энтропии и его будет невозможно угадать. Мы рекомендуем использовать ключ не менее 192 бит. Для генерации такого ключа можно выполнить следующую команду в консоли Ubuntu:

```
$ apg -x 32 -m 32 -a 1
```

Стоит заметить, что данная программа генерирует ключ, где каждый символ представляет собой примерно 6 бит энтропии. Поэтому 32 символа - это примерно 192 бита.

1.2.2 Конфигурация безопасности

Множество настроек существует для Flask приложения. Сюда входят как настройки общего характера, так и настройки относящиеся к безопасности вашего веб приложения. В данной главе мы рассмотрим настройки, относящиеся к безопасности.

https://flask.palletsprojects.com/en/2.0.x/config/

1.2.3 Аутентификация и авторизация

1.2.4 Проверка форм регулярными выражениями

1.2.5 Шифрование данных с SSL

Шифрование потока данных от пользователя до веб сервера является важным атрибутом современного веб приложения: все данные, которые передаются от пользователя до сервера будут зашифрованными (особенно это важно для таких данных, как пароль, данные о кредитной карте, куки файлы, пользовательская история, и множество других). Что бы достичь этого достаточно включить шифрование в Flask приложении следующим образом:

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

@app.route("/")
def hello():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(ssl_context='adhoc')
```

Данный способ будет генерировать самоподписанные сертификаты каждый раз, когда будет стартовать наше веб приложение. Для того, чтобы данный способ работал необходимо установить дополнительную зависимость:

```
$ pip3 install pyopenssl
```

Проблема заключается в том, что при каждом старте приложения генерируется самоподписанный сертификат. Что не совсем удобно. Альтернативой является генерация своего собственного самоподписанного сертификата и использование его при любом старте приложения. Для того, чтобы сгенерировать сертификат достаточно выполнить следующую команду в командной строке Ubuntu:

```
openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem
                                                    -days 365
```

После выполнения команды у вас появится два файла cert.pem и key.pem. cert.pem - это самоподписанный сертификат, а key.pem - это секретный ключ. Срок действия сертификата - 365 дней, а сложность модуля - 4096 бит. Далее настроим наше веб приложение для работы с сертификатом:

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

```
@app.route("/")
def hello():
    return "Hello, World!"

if __name__ == "__main__":
    app.run(ssl_context=('cert.pem', 'key.pem'))
```

Но существует и другая проблема: Самоподписанные сертификаты не нравятся браузерам. При первом открытии приложения, браузер выдаст ошибку, что сертификат не является доверенным и его валидность будет необходимо подтвердить, приняв сообщение об угрозе. Для того, чтобы ваши сертификаты были доверенными браузером можно купить подписанный сертификат у провайдера безопасности. Альтернативой являеться компания LetsEncrypt [?], которая позволяет бесплатно генерировать подписанные и доверенные сертификаты сроком на 90 дней. Для этого достаточно установить библиотеку, настроить DNS имя (у вас должно существовать зарегестрированное доменное имя) и сконфигурировать сервер Nginx на порту 80. Начнем с установки библиотек и приложения certbot:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:certbot/certbot
$ sudo apt-get update
$ sudo apt-get install certbot
```

Далее необходимо сконфигурировать Nginx сервер следующим образом:

```
server {
    listen 80;
    server_name strangebit.io;
    location ~ /.well-known {
        root /vaw/www/certbot;
    }
    location / {
        return 301 https://$host$request_uri;
    }
}
```

И наконец мы можем запросить сертификат для нашего домена:

```
$ sudo certbot certonly --webroot -w /vaw/www/certbot -d strangebit.io
```

Безусловно, домен *strangebit.io* должен быть заменён вашим доменом. После успешного выполнения данной команды, ctrtibot создаст подписанный сертификат в */etc/letsencrypt/live/strangebit.io/fullchain.pem*, а также секретный ключ */etc/letsencrypt/live/strangebit.io/privkey.pem*. Эти ключи

уже можно будет использовать в вашем приложении. Но самое главное браузеры будут доверять таким сертификатам и атаки в виде Man-in-the-middle будут невозможными.

И на последок. Как создать $A+$ SSL сервер? После того, как были сгенерированы сертификаты, остаются ещё пару тонкостей по настройке веб сервера: Первое, необходимо создать параметры Diffie-Hellman, которые будут гарантировать необходимый уровень безопасности, и второе, необходимо запретить небезопасные алгоритмы шифрования.

Для начала создадим параметры для Diffie-Hellamn. Это может занять долгое время, но после выполнения команды у вас будет гаранития того, что обмен ключами будет криптостойким:

```
openssl dhparam -out /etc/security/dhparam.pem 2048
```

И наконец зададим алгоритмы, которые будут доступны при создании SSL соединения:

```
ssl_ciphers 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-
                                        SHA384::ECDHE-RSA-AES256-SHA384
                                        :ECDHE-ECDSA-AES256-SHA384:
                                        ECDHE-RSA-AES256-SHA:ECDHE-
                                        ECDSA-AES256-SHA:DHE-RSA-AES256
                                        -SHA256:DHE-DSS-AES256-SHA:DHE-
                                        RSA-AES256-SHA:AES256-GCM-
                                        SHA384:AES128-SHA256:AES256-
                                        SHA256:AES256-SHA:AES:CAMELLIA
                                        :!DES-CBC3-SHA:!aNULL:!eNULL:!
                                        EXPORT:!DES:!RC4:!MD5:!PSK:!
                                        aECDH:!EDH-DSS-DES-CBC3-SHA:!
                                        EDH-RSA-DES-CBC3-SHA:!KRB5-DES-
                                        CBC3-SHA';
```

Тогда конфигурация Nginx будет выглядеть следующим образом:

```
server {
    listen 443 ssl;
    server_name example.com;
    ssl_certificate /etc/letsencrypt/live/strangebit.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/strangebit.io/privkey.pem
                                        ;
    ssl_dhparam /etc/security/dhparam.pem;
    ssl_ciphers 'ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-
                                        SHA384::ECDHE-RSA-AES256-SHA384
                                        :ECDHE-ECDSA-AES256-SHA384:
                                        ECDHE-RSA-AES256-SHA:ECDHE-
                                        ECDSA-AES256-SHA:DHE-RSA-AES256
```

```
                                                -SHA256:DHE-DSS-AES256-SHA:DHE-
                                                RSA-AES256-SHA:AES256-GCM-
                                                SHA384:AES128-SHA256:AES256-
                                                SHA256:AES256-SHA:AES:CAMELLIA
                                                :!DES-CBC3-SHA:!aNULL:!eNULL:!
                                                EXPORT:!DES:!RC4:!MD5:!PSK:!
                                                aECDH:!EDH-DSS-DES-CBC3-SHA:!
                                                EDH-RSA-DES-CBC3-SHA:!KRB5-DES-
                                                CBC3-SHA';
    ssl_protocols TLSv1.3;
    ssl_session_timeout 1d;
    ssl_session_cache shared:SSL:50m;
    ssl_stapling on;
    ssl_stapling_verify on;
    add_header Strict-Transport-Security max-age=15768000;
}
```

1.3 SQL: Модель данных

1.3.1 Объектно-ориентированный подход к запросам

1.3.2 Миграции

1.4 NoSQL

1.4.1 MongoDB

1.4.2 Cassandra

1.5 Представления

1.5.1 Jinja

1.5.2 Веб формы

1.5.3 Bootstrap

1.6 Контроллеры и бизнес логика

1.6.1 Blueprints: Структурируем большое приложение

1.7 Клиентский код: Vue.js

ГЛАВА 2

Разрабатываем приложение на примере машинного обучения

Литература

[1] Д. Купцов. Python в примерах. 2020. https://github.com/dmitriykuptsov/python_in_examples/blob/master/book.pdf (visited 2021-05-14).