План урока

- Разберемся, нужно ли знать фронтенд бэкенд-разработчику
- Познакомимся с HTML шаблонизатором Jinja и напишем простенькие HMTL странички для нашего API
- Привяжем статический контент к нашему приложению через маунтинг
- Поймем, как привязать фронтенд к бэкенду и познакомимся с CORS
- Научимся принимать файлы от пользователей и сохранять их

Бэкенд + Фронтенд = ❤

Какие бы чувства вы не испытывали к фронтенду: работе с HTML, CSS и JavaScript, что бы вы ни думали о фронтендерах и их способностях к "реальному кодингу", в наше время почти каждый бизнес оформляет свой продукт в интернете через вебсайт и, таким образом, задействует и фронтендеров и бэкендеров.

Код для фронта обычно пишется на фреймворках, например, React.js, Vue.js или AngularJS. Для очень простой верстки с минимумом логики бэкендеры примеряют на себе шкуру фронтендера и самостоятельно пишут простенькие HTML файлы и CSS стили без JS или с его минимальным использованием.

Несмотря на то, что мало кто будет заставлять вас писать и бэкенд и фронтенд, кое-что знать о фронте вам все-таки необходимо. Об этом написано ниже под заголовком **CORS: связываем бэк и фронт**.

Шаблонизатор Jinja2

Как известно, обычная HTML страничка сама по себе статичная. Чтобы изменить в ней какой-то элемент без обновления страницы, нужен JS. Мы JS пока не знаем, поэтому прибегнем к часто используемому решению для отрисовки страниц с динамическим содержанием — шаблонизаторам. Самый популярный шаблонизатор для Python — Jinja2, он используется и в Django, и в Flask, и в FastAPI. Это библиотека позволяет выделять в HTML-коде места, в которые затем будут подгружены данные. Давайте взглянем на пример:

```
<h1>{{ hotel["name"] }}</h1>
```

Если через Jinja2 передать в такой шаблон переменную `hotel = {"name": "Отель Relax"}`, то шаблонизатор превратит наш шаблон в статический файл следующего содержания:

```
<h1>Отель Relax</h1>
```

Мы не будем сильно увлекаться написанием HTML и CSS в этом курсе и лишь взглянем на азы HTML.

CORS: связываем бэк и фронт

Несмотря на то, что от начинающих бэкендеров не требуют знания HTML, CSS и JS, от них тем не менее требуют понимания работы HTTP запросов, отправляемых с браузера в наш API. Дело в том, что по умолчанию наш API не принимает запросы с браузеров. Для настройки доменов (или их IP-адресов), с которых мы можем получать запросы, необходимо сконфигурировать CORS на бэкенде. CORS — это механизм контроля доступа для обращения браузера к внешнему источнику.

Если вы вдруг найдете классное API, которое отдает картинки самых пушистых котят и начнете создавать свое фронтенд-приложение в надежде использовать это API для отображения картинок котиков в красивой рамочке на вашем сайте, вы столкнетесь с ошибкой CORS. Скорее всего создатель API, отдающего пушистых котиков, запретил таким энтузиастам как вы обращаться к нему из браузера, ведь это довольно сильно может эксплуатировать подобный API, если, например, на ваш сайт будет заходить большое количество человек.

> Если я буду обращаться к такому API не из браузера, а через Python, никакого CORS ведь не будет! Почему так?

Количество людей, умеющих обращаться к API через Python или другой ЯП, по отношению к количеству людей, пользующихся браузером, стремится к нулю. Причем люди, обращающиеся к API через Python, понимают, что они делают. А вот те, кто не умеют кодить, а просто листают ленту и смотрят на котят, могут не подозревать, какие уловки мошенники могут соорудить, используя возможность обращаться к стороннему API. Для того, чтобы обезопасить всех интернет-пользователей от возможных кибератак, придуман CORS.

> Как же нам обезопасить наш API?

Мы с вами тоже запретим доступ к нашему API с любого сайта в интернете и дадим доступ лишь домену, на котором крутится наш фронтенд. Фронтенд на JS мы конечно писать не будем, так как это выходит за рамки курса, но для примера на простом приложении на фреймворке React.js я покажу, как настраивается CORS для успешного взаимодействия фронт-бэк.