CORS

CORS (Cross-Origin Resource Sharing) - это механизм, основанный на передаче заголовков и предназначенный для запрета запроса данных с другого источника. Этот механизм обычно не используется при запросе статических файлов (css, js, картинки), он предназначен в основном для использования с Ajax-запросами (например, POST-запрос с фронта на бэк). Использование этого механизма удобнее, чем разрешать запрашивать данные только с того же ресурса, но безопаснее, чем разрешать запрашивать любые данные.

Что значит запрос данных с другого ресурса? Например, вы создали html-страницу и воспользовались для ее верстки стилями из библиотеки Boostrap. У Bootstrap есть удобный CDN - вы можете просто указать ссылку на сервер Bootstrap'а и загрузить оттуда нужный CSS-файл со стилями. При рендеринге вашей страницы, браузер увидит ссылку на внешний ресурс и произведет запрос за нужными файлами. Это и есть Cross-Origin запрос: браузер запросил html-страницу с сайта mywebpage.ru, а потом эта страница запрашивает данные с сайта cdn.bootstrap.com. Любой запрос отличающийся от источника (в нашем случае источник это mywebpage.ru, так как именно с этого домена мы запросили первоначальную страницу) хостом, портом или даже протоколом, является Cross-Origin запросом.

Что такого плохого может произойти без CORS? Например, вы зашли на сайт ukradu.ru. На нем злоумышленик, с помощью XSS-атаки, или просто злобный админ разместили вредоносный скрипт. Он делает простую вещь: отправляет запрос на API банка, который переводит все деньги на счет другого человека (конечно, таких методов в открытом доступе нет, но моделируем самый жуткий сценарий). Ваш браузер видит, что нужно выполнить код злоумышленника и посылает запрос на API. А вместе с запросом уходят куки, которые были поставлены банком, при вашем предыдущем запросе. И банк, отвечая на запрос, думает что общается со своим любимым клиентом, а не с кодом злоумышленика и с удовольствием переводит ваши деньги на чужой счет. Конечно, этот пример немного утрирован - банки обычно не ставят куки, которые могут быть доступны на других сайтах (с помощью аттрибутов domain и same-site).

Более реальный пример: на своем сайте злоумышленик держит код, который посылает запрос на сайт компании, где вы работаете, например, чтобы просто показать картинку, которая доступна только авторизованным пользователям. Если картинка показалась, то значит, что вы авторизованы на сайте компании, а значит вы работаете в этой компании. Теперь не так сложно просто позвонить вам, представиться службой безопасности вашей компании и узнать от вас логин/пароль, то есть использовать социальную инженерию.

Но порой CORS мешает жизни разработчиков, особенно если происходит локальная разработка на вашем компьютере. Вы запускаете фронтенд и бэкенд, но неожиданно фронтенд не может послать запрос на бэк из-за проблем с CORS. Это все потому что, фронт запущен, например на 127.0.0.1:3000, а бэк на 127.0.0.1:8080 - и браузер считает, что был произведен Cross-Origin запрос, порты ведь разные.

Как устроен механизм работы CORS нет смысла рассказывать здесь - материалов много в интернете и тема курса не состоит в изучение уязвимостей. Но стоит сказать, как решить проблему описанную в предыдущем абзаце.

Решение проблемы с CORS **при локальной разработке**

При разработке aiohttp сервера можно использовать библиотеку aiohttp-cors.

Чтобы ее использовать надо при настройке aiohttp.web.Application также настроить aiohttp_cors:

```
from aiohttp import web
import aiohttp_cors

async def handler(request):
  return web.Response(text='hello')

app = web.Application()
# настраиваем aiohttp_cors и разрешаем запрашивать наши методы с любого источника
cors = aiohttp_cors.setup(app, defaults={
    "*": aiohttp_cors.ResourceOptions(
            allow_credentials=True,
            expose_headers="*",
            allow_headers="*",
        )
})
# добавляем наш Route в обработку CORS-запросов
cors.add(app.router.add_post('/method', handler))
```

Конфигурация aiohttp_cors.setup в примере разрешает любые запросы на ваш сервер с любого Origin. То есть, при запросе с *127.0.0.1:3000* на *127.0.0.1:8080/method* теперь не возникнет CORS-ошибки. Только надо не забывать добавлять нужные Route в объект *cors*.

Если вы разрабатывает просто API-сервер и тестируете его с помощью Postman, то вам **не** нужно использовать aiohttp-cors.