План урока

- Поймем разницу между аутентификацией и авторизацией
- Познакомимся с аутентификацией через JWT
- Напишем первые зависимости через Depends для извлечения данных (cookies/headers) из запросов пользователей
- Напишем эндпоинты для регистрации, входа и выхода пользователя
- Разберемся с HTTP кодами ответа (200, 401, 500 и др.) для грамотной обработки ошибок и их отдачи пользователям

Аутентификация / Авторизация

Эти термины путаются очень часто. Давайте запомним раз и навсегда, чем отличается аутентификация и авторизация.

Аутентификация — проверка, что вы являетесь тем, за кого себя выдаете. Например, вы ввели на сайте ваш логин Pr0ger. Чтобы подтвердить системе, что вы являетесь этим самым Pr0ger'ом, вы вводите пароль аккаунта. Если такая пара логин + пароль есть в базе данных, то процесс аутентификации завершен, так как сервер распознал вас как одного из уже существующих пользователей в базе.

Авторизация — проверка наличия у вас прав при обращении к какому-либо эндпоинту. Она происходит каждый раз(!) при обращении к любому защищенному эндпоинту (который требует каких-либо прав, даже самых минимальных).

> Почему каждый раз? Разве мы не можем один раз проверить пользователя, а все последующие запросы не проверять его?

На протяжении всего курса я стараюсь следовать стилю REST при построении API. API, соблюдающее часть требований стиля REST называется RESTful API, что в переводе значит REST-подобное API. Одним из требований к архитектуре REST является отсутствие состояния. Именно поэтому мы каждый раз проводим процесс авторизации.

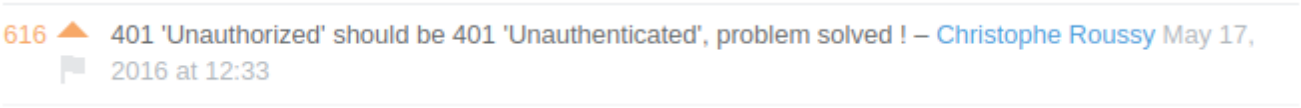
> Почему процесс аутентификации не проводим каждый раз? В чем отличие от авторизации?

Я забыл упомянуть, что после аутентификации клиент получает закодированный токен (набор букв и цифр по типу ejSs28sAkzmfa$TAp10nEl0V3), который хранит у себя. Если речь идет про браузер, то такой токен обычно хранится в локальном хранилище (local storage) или кукис (cookies). Токен каждый раз передается вместе с запросом, поэтому аутентификация каждый раз не проводится.

> Слышу, что другие люди часто используют слово авторизация, когда говорят об аутентификации. Почему так?

Это нормально произносить *авторизация* вместо *аутентификация*, потому что так легче и быстрее, но важно ориентироваться в этих понятиях и не путать их.

Что забавно для меня, так это то, что код 401 Unauthorized (не авторизован) на самом деле должен называться Unauthenticated (не аутентифицирован). Можете взглянуть на этот ответ на stackoverflow: https://stackoverflow.com/a/6937030/18406890



Перевод: *401 "Не авторизован" должен называться 401 "Не аутентифицирован", проблема решена!*

Таким образом, код 403 Forbidden (запрещено) должен отправляться уже аутентифицированным пользователям, которых распознал сервер, а код 401 Unauthorized должен отправляться, когда сервер не может распознать пользователя (т.к. нет нужного токена для этого).

Запоминалка: 401 - юзер не залогинен, 403 - юзер залогинен, но нет прав на данный эндпоинт

JWT токен

В курсе рассматривается работа с JWT токеном. Это один из самых популярных видов добавления аутентификации в наше приложение. Мы не погружаемся в эту тему глубоко, так как она довольно сложная и Junior и Middle специалистов редко подпускают к работе над сервисами аутентификации и авторизации. Давайте лишь вкратце перечислим нюансы при использовании JWT:

- Пример типичного JWT токена: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`
- JWT токен состоит из трех частей: заголовка, данных и подписи
- JWT токен не зашифрован, а лишь подписан. Это значит, что все данные внутри токена могут быть прочитаны кем угодно
- JWT токен создается и подписывается на сервере с помощью секретного ключа. Невозможно подделать токен без знания секретного ключа
- JWT токен всегда имеет ограниченный срок жизни
- JWT токен хранится у клиента и отправляется вместе с каждым запросом
- Для реализации аутентификации через JWT необходимо отправлять клиенту два токена: короткоживущий и долгоживущий, которые клиент сохранит у себя внутри браузера (либо в отдельных переменных, если кто-то обращается к нашему API через скрипт на каком-нибудь языке программирования)
- Короткоживущий токен (токен доступа) обычно действует 15-30 минут, долгоживущий (токен обновления) действует от нескольких дней до нескольких месяцев

- Короткоживущий токен хранится только на клиентской стороне, в то время как долгоживущий хранится и на клиенте и на сервере (обычно в базе данных)
- При истечении короткоживущего токена клиент отправляет долгоживущий токен и получает новую пару короткоживущего и долгоживущего токенов
- При краже короткоживущего токена у пользователя мошенник может выдавать себя за пользователя, пока не истечет время действия токена
- При краже долгоживущего токена у пользователя мошенник может выдавать себя за пользователя до тех пор, пока этот долгоживущий токен не будет признан невалидным этим же пользователем или поддержкой сайта

Как видите, нужно учитывать довольно много факторов. В курсе этой теме посвящено не очень много времени ввиду сложности материала.

Для глубокого понимания горячо рекомендую к прочтению эту статью: https://gist.github.com/artemonsh/34345edb40d9097f94bd54aa4b8313f6. Это мой форк (без изменений) чудесного русскоязычного материала по JWT токенам. Для полного понимания предлагаю прочитать материал 2-3 раза.