**Занятие sessions ↔ Express.js**

**Тема: аутентификация сессиями**

Идентифицировать X – это означает распознать объект в качестве X, т.е. выяснить, что объект – это элемент X некоторого множества. Т.е. соотнести признаки объекта с признаками, которые служат для включения X в некоторое множество.

Например, чтобы доказать, что некто является клиентом банка, некто предъявляет паспорт. Сверив лицо предполагаемого клиента с фотографией в паспорте и сверив номер паспорта с записью в базе данных, сотрудник банка идентифицирует его или её как клиента. Паспорт бывает достаточен для снятия денег со счёта, но чтобы, к примеру, заблокировать карту по телефону, нужно назвать ещё и кодовое слово.

Всё это – вопросы точности идентификации. С одной стороны, паспорт однозначно представляет гражданина страны. С другой стороны, можно украсть паспорт. С ещё другой стороны, подделать можно и внешность. Что же идентифицируется? Уникальная личность, которая определяется чем?

В общем, это довольно условно. Для простоты – компьютерная система оперирует моделью пользователя, представленной на базовом уровне логином и паролем.

Если мы знаем, что два разных человека пользуются одним и тем же логином и паролем, то для системы в рамках такой модели это один и тот же пользователь.

Идентифицировать – это значит, грубо говоря, найти пользователя в множестве пользователей – принять его или её заявку на вход в систему, а аутентифицировать – значит действительно подтвердить, что это тот самый пользователь, за которого кандидат себя выдаёт. Эти два акта тесно связаны и разделить их нелегко, да и нет необходимости. Что же касается авторизации, то это отнесение аутентифицированного пользователя к той или иной группе в аспекте *прав*, предоставление ему личного аккаунта / кабинета сообразно с этими правами.

Двухэтапная или двухфакторная аутентификация – такая, при которой используются два идентифицирующих фактора из разных категорий (их три: что **только этот** пользователь **знает**, что **имеет** и чем **является**). Например, пользователь **знает** пароль и **имеет** мобильный телефон. Значит, после успешного ввода пароля на телефон придёт сообщение с уникальным краткосрочным кодом. К третьей категории относятся ДНК, отпечатки пальцев и т.п.

Логин отсюда надо исключить, т.к. это общедоступная информация. Её знает не только пользователь. Грубоая аналогия: лицо (логин) и паспорт (пароль). Это также можно сопоставить с шифрованием двумя ключами (публичный и секретный).

Цель умножения числа факторов – снизить вероятность идентификации, которая приведёт к неправильной аутентификации. Но вероятность того, что даже самые продуманные меры не помогут, всё равно ненулевая.

Выражаясь осторожно, закрытая часть сайта/приложения – это такая, доступ к которой можно получить только после успешной аутентификации. Т.е. приватный раздел увидит только тот, кто ввёл существующий логин и верный соответствующий этому логину пароль.

В Express это реализуется, конечно же, с помощью middleware.

Идея в том, что обработчик маршрута к закрытому разделу не сразу выдаёт то, что там находится, а только если будет успешный ответ от соответствующего middleware.

Т.е., например, все маршруты под маршрутом /users или /profile или /cabinet должны выдаваться только через этот middleware.

Сессия – это состояние «включена аутентификация», которое может быть программно реализовано по-разному. Но начинается всё с того, чтобы в это состояние прийти, т.е. аутентифицироваться, т.е. «залогиниться».

Для этого логично выделить маршрут типа /login или /signin

Этот маршрут должен выдавать форму, данные из которой идут, например по маршруту /login/check

|  |
| --- |
| наглядная имитация с помощью адресной строки (GET)  клиент: <http://kodaktor.ru/g/session_get> (реальный случай <http://kodaktor.ru/g/session_post>) |
| app.get('/login/check/:login/:pass', (req, res)=>{  //1. ищем логин в 'u' (симулякр DB в виде users.json)  **if** (req.params.login **in** u) {  //2. проверяем пароль  **if** (u[req.params.login] **==** req.params.pass) {  req.session.auth **=** 'ok';  req.session.login **=** req.params.login;  res.redirect('/profile'); //осуществим авторизацию  } **else** {  **console**.log('неверный логинь или пароль');  res.redirect('/signin'); //неверный пароль – обратно в форму!  };  } **else** {  **console**.log('неверный логинь или пароль');  res.redirect('/signin'); //не найден пользователь – обратно в форму!  };  }); |

который должен обслужить приём данных из неё. В случае успешной сверки реквизитов сессия инициализируется, т.е. грубо говоря создаётся объект, хранящий некий идентификатор этой сессии (session ID).

Таким образом модель «логин-пароль[-…]\*» превращается в session ID, ужимаясь до одного объекта.

Далее,

npm install express-session

session = require('express-session')

|  |
| --- |
| package.json |
| {  "name": "express-sessions-demo",  "version": "1.0.0",  "description": "a session express app",  "main": "index.js",  "engines": {  "node": "6.0.0"  },  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1",  "start": "node index.js"  },  "repository": {  "type": "git",  "url": "https://github.com/gossoudarev/webteach/tree/master/express-sessions-demo"  },  "keywords": [  "sessions",  "express"  ],  "author": "EliasGoss",  "license": "ISC",  "dependencies": {  **"body-parser": "1.15.0",**  "express": "4.13.4",  **"express-session": "1.13.0",**  "request": "2.72.0"  } } |