

6. Добавление авторизации и пользовательских сессий

В результате выполнения лабораторной работы на разрабатываемом веб-сайте должны быть реализованы механизмы авторизации через сторонних поставщиков услуг и аутентификации на целевом ресурсе.

Контрольный срок сдачи работы: 14-ая неделя обучения.

Для выполнения данной лабораторной работы вам необходимо реализовать в ранее созданном приложении модуль, который бы отвечал за предоставление прав в рамках пользовательской сессии внутри контроллеров, спроектированных в рамках предыдущей лабораторной работы.

Авторизация на вашем ресурсе позволяет пользователю получить доступ к изменению настроек своей учетной записи, интерфейса взаимодействия с системой, паролей, типовых операций, внесение изменений в систему. До выполнения авторизации, посетитель сайта может использовать лишь ограниченный набор функций.

Для удовлетворения всех стандартов безопасности, которые существуют на сегодняшний день предлагается вместо того, чтобы разрабатывать свой механизм авторизации, воспользоваться поставщиками таких услуг как Authorization as a Service/as a Platform. Зачем пользоваться сторонним решением для того чтобы провести аутентификацию пользователя? Вы не можете обеспечить надлежащее управление данными и безопасность. Иными словами, если вы не Папа Джонс, то изготовление пиццы с нуля может пойти не так.

В зависимости от того, насколько сильную кастомизацию UI/UX вы хотите, предлагаю рассмотреть следующих поставщиков услуг:

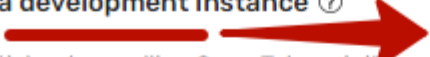
- [Auth0](#) — это платформа, предоставляющая готовые решения для разработки сервисов любого уровня сложности. Auth0 поддерживается командой, стоящей за разработкой JWT (JSON Web Token). Полностью проприетарный продукт. Предоставляют как Backend для авторизации, так и настраиваемую форму для ввода логина/пароля со своей стороны. Существуют готовые рецепты для интеграции с приложениями написанными на NodeJS. Но на данный момент не предоставляют свой сервис для клиентов регистрирующихся из России.
- [Super Tokens](#) — это альтернатива Auth0 с полностью открытым исходным кодом, которая позволяет настроить аутентификацию менее чем за 30 минут. Есть возможность воспользоваться Managed Infrastructure, т.е. сам сервер авторизации (Identity Provider) доступен вам как облачное решение (используют AWS для размещения своих серверов). Комплексное решение, включающее вход в систему, регистрацию, управление пользователями и сессиями без сложностей протоколов OAuth.

- [Google FireBase Auth](#) - Это система аутентификации на основе токенов, которая обеспечивает легкую интеграцию с большинством платформ. в свое приложение авторизацию можно добавить с помощью пакета SDK. Есть возможность использовать как свою форму для входа так и стандартную, предоставляемую самим сервисом.



В рамках данной лабораторной не столь важно каким именно поставщиком услуг вы воспользуетесь для того чтобы авторизовывать пользователей, но я рекомендую остановить свой выбор на системе SuperTokens т.к. здесь вам необходимо будет реализовать все необходимые классы внутри вашего приложения самим, что добавит понимания о том как именно происходит сам процесс авторизации!

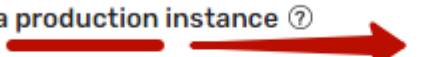
Далее будут расписаны требования к реализации модуля авторизации и аутентификации на примере сервиса SuperTokens, но эти требования характерны и в случае использования других сервисов.

После регистрации и выборе типа размещения вашего авторизационного сервиса (Managed Infrastructure) вам будет сгенерирована конфигурация для тестовой (Development) среды и для боевой (Production) среды:



Connecting to a development instance ?  localhost

Put these credentials when calling SuperTokens init function on your backend:

connectionURI:	<input type="text" value="https://76369ee1ba9411ecafb627..."/>	
API key:	<input type="text" value="Y25PGS4zJVx7-rCdVlu7-SLEfhs=H"/>	

Connecting to a production instance ?  Heroku

Put these credentials when calling SuperTokens init function on your backend:

connectionURI:	<input type="text" value="https://3caa2601ba9511ecafb627..."/>	
API key:	<input type="text" value="vuAGdhWfVKNHVH9nJrZSbt99KC..."/>	

Всю конфигурацию необходимо вынести в переменные среды окружения. Сделать это можно таким же образом как и для строки подключения к базе, через команду ``heroku config:set SecretApiKey=VuAGdh....`` в терминале используя Heroku CLI.

Весь код для подключения модуля авторизации необходимо вынести в отдельный инфраструктурный слой согласно лучшим практикам DDD.

После того как вы вынесите необходимые данные для подключения рекомендуется ознакомиться с теми подходами, которые предоставляются тем или иным поставщиком услуг. Вы можете выбрать авторизацию по ссылкам на почту, с помощью сторонних ресурсов либо классический вариант, через логин и пароль.

Создайте отдельный модуль внутри вашего приложения, это можно сделать используя команду `nest g module auth`

Внутри модуля **необходимо реализовать** возможность его конфигурации в момент выполнения программы путем **передачи конфигурации при регистрации модуля**. Такой подход в Nest называется [динамические модули](#). т.е. конфигурационные значения из переменных окружения вычитываются на старте приложения и передаются в статический метод вашего динамического модуля, для получения сконфигурированного экземпляра, который будет доступен непосредственно при инъекции зависимостей в другие ваши модули.

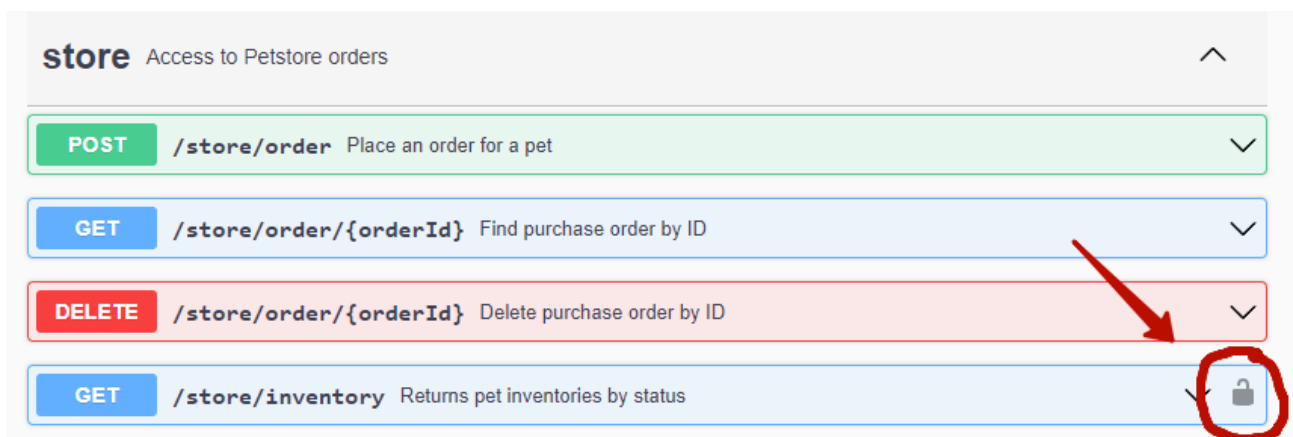
Далее, вам так или иначе необходимо будет разграничить какие ресурсы вы готовы предоставлять всем пользователям, а какие только тем, кто является авторизованным пользователями. Для этого рекомендуется воспользоваться Nest'овскими [Guard](#)'ами. Вы можете использовать уже готовые шаблонные AuthGuard'ы из npm пакета PassportJS, каким либо готовым из SDK пакета того поставщика услуг при наличии, либо написать свой Guard, заточенный под ваш конкретный вариант использования.

Ваш AuthGuard должен реализовывать интерфейс **CanActivate** и в зависимости от того, авторизован ли пользователь, возвращать true либо false. Методы контроллера, требующие авторизацию необходимо декорировать в помощью `@UseGuard(BашAuthGuard)`

У guards есть единственная ответственность. Они определяют, будет ли данный запрос обработан обработчиком маршрута или нет, в зависимости от определенных условий (таких как разрешения, роли, ACL и т.д.), существующих во время выполнения. Это часто называют авторизацией. Аутентификация, с которой обычно взаимодействует AuthGuard обрабатывается через middleware в традиционных приложениях Express. Middleware - отличный выбор для аутентификации, поскольку такие вещи, как проверка токенов и прикрепление свойств к объекту request, не сильно связаны с конкретным контекстом маршрута (и его метаданными). Они, как и фильтры исключений, pipes и interceptors, предназначены для того, чтобы вы могли вмешаться в логику обработки в нужный момент цикла запроса/ответа, Это помогает сохранить ваш код цельным и декларативным.

Как вы поняли, в случаях, когда пользователь запрашивает ресурс, доступный только авторизованным пользователям, но при этом, в текущей сессии процесс идентификации ещё не был произведён, необходимо переадресовать пользователя на форму входа. Для этого необходимо **реализовать** свой **Middleware**, который бы обрабатывал этот сценарий. Для подключения такого обработчика к конкретному контроллеру рекомендуется воспользоваться Nest'овским подходом [Middleware Consumer](#) при регистрации того или иного контроллера внутри модуля.

После того как вы добавите Guards & Middleware, необходимо также обозначить в документации к вашему API, какие именно методы требуют авторизации и какая именно схема используется для этого. SwaggerModule поддерживает несколько [стандартных типов схем](#). Выберите ту схему авторизации, которую предоставляет ваш поставщик услуг и [декорируйте](#) те контроллеры, куда вы только что добавляли указание использовать AuthGuard. Если этот шаг сделан верно, то на методах, требующих авторизацию появится значок замочка. см пример ниже:



После этого, т.к. вы уже знаете как именно передается авторизационный токен (в cookie или в конкретных Header'ax), необходимо настроить [CORS Policy](#) внутри вашего приложения в файле main.ts:

```
app.enableCors({ options: {  
  origin: ['https://your-application.herokuapp.com'],  
  allowedHeaders: ['content-type', ...supertokens.getAllCORSHeaders()],  
  credentials: true,  
});
```

Если вы используете не SuperTokens, то набор разрешённых Header'ов должен содержать все заголовки, отличающиеся от [стандартных](#).

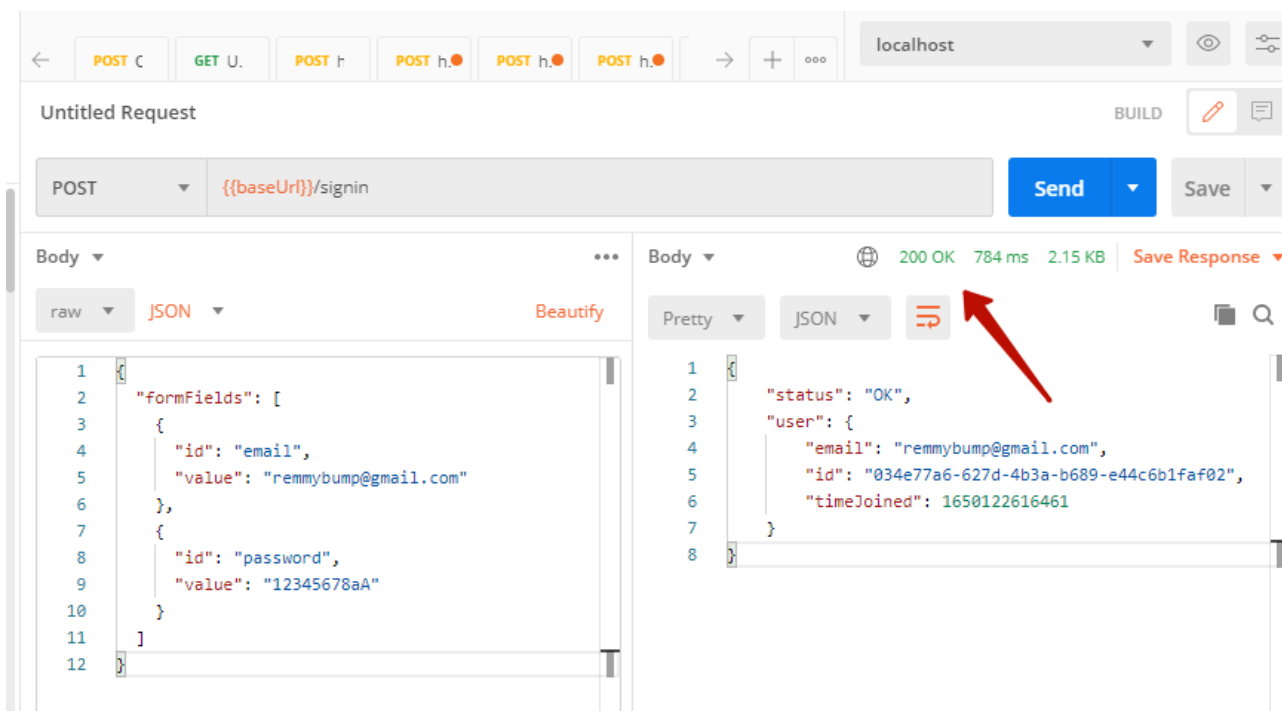
К этому моменту считаем что у нас есть вся необходимая обвязка для проверки токена, осталось его выпустить. Если ваш поставщик услуг авторизации не предоставляет готового frontend'а с логин формой, необходимо сверстать её самостоятельно и послать запрос к авторизационному сервису.

В случаях если вы используете OAuth 2.0, то попросите соответствующие Grant'ы у Identity Provider'а и обменяйте полученный код на авторизационный токен.

В случаях если вы не используете OAuth 2.0, как например в SuperTokens, просто обратитесь на Authorization Endpoint с указанием схемы авторизации.

Если форма с логином ещё не готова, то для проверки рекомендую воспользоваться HttpClient'ом например Postman:

Делаем запрос на авторизационный сервис:





Убеждаемся что получен валидный ответ и юзер прошёл авторизацию, получаем Cookies (в моём случае я выбрал авторизацию через Cookie).

В соседней вкладке смотрим что мы получили (см. скриншот ниже):

Body Cookies Headers (15) Test Results		200 OK 784 ms 2.15 KB Save Response ▼
KEY	VALUE	
X-Powered-By ⓘ	Express	
Vary ⓘ	Origin	
Access-Control-Allow-Credentials ⓘ	true	
front-token ⓘ	eyJ1aWQoOilwMzRINzdhNi02MjdkLTRiM2EtYjY4OS1INDRjNmIjZmF...	
Access-Control-Expose-Headers ⓘ	front-token, id-refresh-token	
Set-Cookie ⓘ	sAccessToken=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsInZlcnNpb24i...	
Set-Cookie ⓘ	sRefreshToken=6ckYVs%2F5tdAeiwVMIO%2BLpH%2F2sVE6RLC83...	
Set-Cookie ⓘ	sIdRefreshToken=fa2e63f5-6796-4b41-8539-49bb3d460a8a; Path=/...	
id-refresh-token ⓘ	fa2e63f5-6796-4b41-8539-49bb3d460a8a;1658863485891	
Content-Type ⓘ	application/json; charset=utf-8	
Content-Length ⓘ	125	
ETag ⓘ	W/"7d-f/46qwqJg5o3IJLEvq5luQGadh8"	
Date ⓘ	Sun, 17 Apr 2022 19:24:45 GMT	
Connection ⓘ	keep-alive	
Keep-Alive ⓘ	timeout=5	

Был получен AccessToken, т.е. то что нам нужно для того обращаться в наш ресурс.

Внутри AccessToken cookie лежит JWT токен с идентификатором пользователя. Для отладки можно воспользоваться сервисом <https://jwt.io>


Debugger Libraries Introduction Ask
Crafted by  auth0

Encoded
PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsInZlcnNpb24iOiJyIn0=.eyJzZXNzaW9uSGFuZGx1IjoimTQ2MjUwNmMtZTczS00MWIxLWE4MWQtM2IzYTg2MWM2MDc0IiwidXN1cklkIjoimDM0ZTc3YTYtNjI3ZC00YjNhLWI2ODktZTQ0YzZiMWZhZjAyIiwicmVmcVzaFRva2VuSGFzaDEiOiJkMzg5MjhmOjc2MjY0Yjc5MzZhNzQ4MGZnNzY3NzZmYWVzODFlZTAzMzFMDIzNmQ1MmE3ZmNlYWVhNWRjZjAxIiwidXN1ckRhGEiOnt9LClleHBpcn1UaW11IjoxNjUwMjI3MDg1ODkyLCJ0aW1lQ3JlYXR1ZCI6MTY1MDIyMzQ4NTg5MiwibG1ydCI6MTY1MDIyMzQ4NTg5Mn0=.FE8qZDQN1tgNUezH%2B%2BZtpr6pP93hAi2Vvm96owsDGKoex396xyV3CwEHWIL1exjusec6xzv735Rxi1vetsYARbcQ1BfpANZ1ZvBucsbmv1rM6hmc2EMisniLEoeXKH%2FX4b1TLUU%2Fb6rwb6e7ahdha0qgEzSyZqj4tMqCMlkCvuy67kae77upXKzw1xlJe%2F0%2Bf31nFWt0cc0vTxcraN08dNve3
```

Decoded
EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "version": "2"
}
```

PAYLOAD: DATA

```
{
  "sessionHandle": "1462506c-e73e-41b1-a81d-3b3a861c6074",
  "userId": "034e77a6-627d-4b3a-b689-e44c6b1faf02",
  "refreshTokenHash1": "d38928f976264b7936a7480b37f776fab381ee0331e0236d52a7fcaeaa5dcf01",
  "userData": {},
  "expiryTime": 1650227085892,
  "timeCreated": 1650223485892,
  "lmrt": 1650223485892
}
```

Как видим, у пользователю был выдан GUID идентификатор - 034e77a6-627d-4b3a-b689-e44c6b1faf02

В дальнейшем всё что нам нужно сделать это вычитать из токена этот userId и связать с тем пользователем который есть у вас в базе данных.

P.S. SuperTokens уже имеет свою обязанку чтобы это делать, см. документацию.

```
@UseGuards(AuthGuard)
getSecret(@Session() session: SessionContainer): object {
  const userId = session.getUserId();
  const token = jwt_decode(se
  return {
    message: JSON.stringify(t
  }
}
```

SessionContainerInterface.getUserId(
 userContext?: any): string

supertokens-node

Если ваше веб-приложение поддерживает аутентификацию и разделение пользователей по ролям, то как второй шаг вы можете выдать роли, соответствующие вашему пользователю уже после прохождения авторизации. Для этого, [согласно документации](#), вам также нужно будет декорировать необходимые методы контроллера с помощью Guard'ов с указанием той или иной роли, необходимой для доступа к вашему методу.

После реализации всех шагов, обновите ваши вьюшки и подкладывайте в контекст информацию о пользователе, чтобы в зависимости от того, вошёл пользователь или нет, в шапке вашего сайта предлагалось либо войти, либо отображалась информация о пользователе.

Почта Картинки



Войти

Почта Картинки



Аккаунт Google
Roman Makarevich

Удачи :)