

## Создание веб-приложения с простой аутентификацией на основе сессий

Цель: создать серверное приложение, которое получает массив объектов, каждый из которых имеет свойство `login`, из базы данных `readusers`, упоминавшейся в предыдущих темах и использует шаблон `pug` для вывода в веб-страницу списка этих логинов при условии ввода верных логина и пароля, которые хранятся по указанному адресу.

Следует взять за основу код по адресу `kodaktor.ru/x`.

При попытке перейти по маршруту `/users` должна происходить проверка того, что сессия установлена, и только в этом случае должен быть показан список логинов; иначе должен быть осуществлён переход на форму для ввода логина и пароля.

### Часть I.

1. Создав пустую папку и перейдя в неё, инициализируйте проект:

```
yarn init -y
```

2. Загрузите базовый код

```
curl -L kodaktor.ru/x -o index.js
```

3. Добавьте зависимости

```
yarn add express pug mongoose body-parser express-session
```

для установки

- `express`
- шаблонизатора `pug`
- ODM `Mongoose`;
- компонента `Express`, отвечающего за обработку данных, посланных методом `post`
- компонента `Express`, отвечающего за обработку сессий

4. Добавьте в проект файлы `conn.js` и `User.js` из предыдущей темы, причём `User.js` поместите в папку `models` и скорректируйте путь в первой строке файла `User.js`

```
const { Schema, model } = require('..../conn');
```

5. Создайте папку `views` для хранения представления

6. Раскомментируйте строку

```
.set('view engine', 'pug')
```

7.

выполните

```
curl -L 'kodaktor.ru/g/session_pug' -o './views/login.pug'
```

для получения файла с шаблоном страницы для ввода логина и пароля

и добавьте маршрут /login после подключения маршрутизатора

```
.get('/login', r => r.res.render('login'))
```

#### 8. Затребуйте зависимости

```
const bodyParser = require('body-parser');
const session = require('express-session');
const { u: User } = require('./models/User');
```

#### 9. Убедитесь, что подключение к БД работает

Для этого можно **временно** создать маршрут

```
.get('/password/:login', async r => {
  const { login } = r.params;
  const result = await User.findOne({ login });
  r.res.send(result ? result.password : 'Такого логина нет!');
})
```

и проверить работу этого маршрута

```
curl localhost:4321/password/ego@yandex.ru
selfish
curl localhost:4321/password/ego@yandex.r
Такого логина нет!
```

(<https://github.com/GossJS/xs/blob/mongoread/index.js> – этот этап)

### Часть II.

#### 10. Добавьте middleware парсера post-запросов и сессий в приложение: (например, после static)

```
.use(bodyParser.json())
.use(bodyParser.urlencoded({ extended: true }))
.use(session({ secret: 'mysecret', resave: true, saveUninitialized: true })))
```

(именно .use(session...) создаёт куку connect.sid, обеспечивая **механизм** сессий, т.е. автоматическое создание объекта req.session и «синхронизацию» его с данными, которые хранятся в кукке)

после чего добавьте маршрут для обработки POST-запросом (если добавляли временный проверочный маршрут на предыдущем шаге, то вместо него) и имитацию закрытого маршрута:

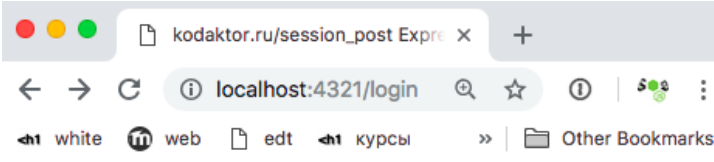
```

.post('/login/check/', async r => {
  const { body: { login } } = r;
  const user = await User.findOne({ login });
  if (user) {
    if (user.password === r.body.pass) {
      r.session.auth = 'ok';
      r.session.login = login;
      r.res.send('Вы авторизованы. Доступен закрытый маршрут!');
    } else {
      r.res.send('Неверный пароль!');
    }
  } else {
    r.res.send('Нет такого пользователя!');
  }
})
.get('/profile', r => r.res.send(r.session.login))

```

Теперь при переходе на маршрут /login мы сможем ввести логин и пароль, помня о том, что записано в БД.

Попробуйте разные комбинации, чтобы убедиться, что алгоритм работает верно.



**Введите логин и пароль!**

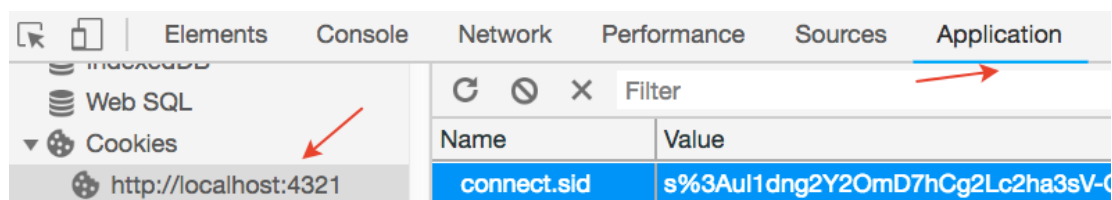
ego@yandex.ru

пароль

**Залогиниться**

В случае успешного ввода при переходе на /profile мы будем видеть логин (электронную почту) в течение жизни сессии; иначе будет пустота, т.к. req.session.login будет undefined, и браузер получит содержимое длины 0.

11. Залогиньтесь с разных браузеров и убедитесь, что после успешной «авторизации» вы будете получать разные ответы по маршруту /profile – важно понять, что express-session не хранит все эти данные у себя, а хранит только идентификаторы сессий. Получив запрос /profile, express-session проверяет наличие куки и если там находится один из идентификаторов, которые сохранены, то из куки считываются данные и превращаются в поля объекта req.session, а иначе объект req.session станет пустым.



Если вручную стереть всю эту куку или её значение, то, опять-таки, объект req.session у данного запроса станет пустым.

Можно сказать, что req.session – это такой аналог ODM, который рассматривался ранее. Когда мы пишем req.session.auth = 'ok' – то через эту инструкцию создаём запись, которая при следующем же запросе (если кука не стёрта) «вернётся» в этот объект.

<https://github.com/GossJS/xs/blob/session1/index.js>

Теперь реализуем удобный способ создания закрытых маршрутов. Можно каждый раз в очередном обработчике маршрута писать что-то вроде if (req.session.auth === 'ok') ... – а можно воспользоваться основным основ Express – механизмом middleware, написав одну функцию, которая будет решать, разрешён ли доступ к маршруту. Назовём её checkAuth.

### Часть III.

12. Добавьте функцию checkAuth промежуточного программного обеспечения (конвейера, middleware) для обработки защищённых маршрутов и защите маршрут /profile ею:

```
const checkAuth = (r, res, next) => {
  if (r.session.auth === 'ok') {
    next();
  } else {
    res.redirect('/login');
  }
};
```

```
.get('/profile', checkAuth, r => r.res.send(r.session.login))
```

Когда приходит запрос на маршрут `/profile`, благодаря размещению ссылки на функцию после шаблона маршрута, в конвейер обработки добавляется наша функция и получает тот же самый объект запроса (`r`), который был создан для этого запроса. В данном случае она проделывает наипростейшую вещь: если поле `auth` имеет значение `ok` (что по нашей внутренней договорённости означает успешную авторизацию), то функция позволяет конвейеру работать с запросом дальше, вызывая `next()` – а иначе переадресует браузер на страницу с логином.

```
curl http://localhost:4321/profile
Found. Redirecting to /login
```

```
HTTP/1.1 302 Found
Location: /login
```

```
https://github.com/GossJS/xs/blob/session2/index.js
```

### Задания

Добавьте маршрут `/logout` для прекращения сессии

Добавьте закрытый маршрут `/users`, а к нему создайте шаблон для красивого отображения списка всех пользователей в виде таблицы `логин – пароль`

Поместите всё получившееся в репозиторий и оставьте ссылку в форме. Нужно, чтобы приложение заработало по команде `npm start` или `yarn start`