

Москаleva Ю.П.

ПРОГРАММИРОВАНИЕ:
УЧЕБНЫЙ ПРОЕКТ НА JAVASCRIPT

УЧЕБНОЕ ПОСОБИЕ

Симферополь
2024

СОДЕРЖАНИЕ

1. Введение	5
2. Сервер на express	6
2.1. Простейший сервер на express	6
2.2. Обработка post запросов	12
2.3. Стартовый проект express	17
2.4. Анализ кода стартового проекта	19
2.5. Выполнение кода JavaScript	21
3. MongoDB	23
3.1. Консоль mongo	23
3.2. Команда FIND	26
3.4. Упражнения для закрепления правил синтаксиса	33
4. Маршрутизаторы и шаблоны стартового express проекта	34
4.1. Стартовый проект с шаблонизатором ejs	34
4.2. Маршрутизатор	37
4.3. Создание шаблона	39
4.4. Установка и подключение шаблонизатора ejs-locals	43
5. Навигация	47
5.1. Подключение Bootstrap	47
5.2. Навигационное меню	51
6. Модуль mongodb, подключение базы данных	54
6.1. Подключение базы данных	54
6.2. Создание модуля данных	58
6.3. Посев данных приложения	61
7. Модуль mongoose, создание моделей данных	63
7.1. ORM	63
7.2. Создание схемы	67
7.3. Создание модели данных приложения	69
8. Отображение данных в браузере	71
8.1. Подключение базы данных	71
8.2. Обработка параметра в адресе	73
8.3. Извлечение данных из базы	75
8.4. Переключение адресов и чистка кода	77
9. Cookie и Session	79

9.1. Установка модуля express-session и настройка cookie	79
9.2. Команда записи в cookie	82
9.3. Сохранение session в MongoDB	85
9.4. Создание счетчика посещения страниц сайта	87
9.5. Глобальная переменная для навигации	89
10. Аутентификация	93
10.1. Создание страницы регистрации	93
10.2. Создание модели User	98
10.3. Подготовка данных для передачи на сервер	102
10.4. Логика пользователя	104
10.5. Глобальная переменная user	107
10.6. Обработка ошибки аутентификации	112
10.7. Функциональность logout	114
10.8. Закрытие страниц сайта для незалогиненного пользователя	
	116

1. ВВЕДЕНИЕ

Учебное пособие посвящено изучению языка программирования JavaScript. Методологически, изучение основ JavaScript, реализуется в рамках разработки учебного проекта. Стек технологий, выбранный для реализации учебного проекта: Node.js, Express.js, MongoDB, EJS.

Что такое Node.js? Для получения корректного ответа на этот вопрос можно обратиться к официальному сайту <https://nodejs.org> :

«Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.»

То есть Node.js это среда выполнения JavaScript. JavaScript имеет две среды выполнения. Исторически первая среда выполнения JavaScript – это браузер. С помощью JavaScript веб-страницам придается интерактивность. Вторая – Node.js, которая поставляется с большим количеством модулей для разработки web-сервера.

Express.js – один из модулей Node.js в котором реализован многочисленный функционал обмена данными с браузером.

MongoDB – база данных. Использование базы данных при разработке учебного проекта делает учебный проект хорошей практикой современного программирования. Выбор именно MongoDB обусловлен прежде всего удобными и гибкими модулями для работы Node.js с MongoDB.

EJS (Embedded JavaScript) – это простой шаблонизатор, который удобно использовать с Node.js и Express.js. Шаблонизаторы позволяют структурировать веб-страницы. В учебном проекте простота EJS позволяет рассмотреть роль шаблонизаторов в разработке веб-приложений.

Выбранный стек технологий: Node.js, Express.js, MongoDB и EJS позволяет понять базовые составляющие процесса разработки проекта.

2. СЕРВЕР НА EXPRESS

2.1. ПРОСТЕЙШИЙ СЕРВЕР НА EXPRESS

Первый шаг – это установка Node.js. Установка Node.js зависит от операционной системы и как для любого процесса инсталляции, пошаговая, подробная инструкция ищется, по ключевым словам (например, «установка Node.js»), в интернете.

Устанавливаем Node.js и открываем консольное приложение компьютера (для OS Window cmd ...).

Для проверки, установлен ли Node.js, в консольном приложении выполним команду:

```
$ node -v  
v22.8.0
```

Это последняя версия (latest) на сентябрь 2024

«Выполнить команду» значит написать команду в консоли и нажать клавишу *Enter*.

Команда *node -v* это запрос версии Node.js.

Разработку проекта начнем с создания папки проекта. Место расположения папки не является важным. Создадим папку проекта:

```
$ mkdir folder  
$ cd folder  
$ pwd
```

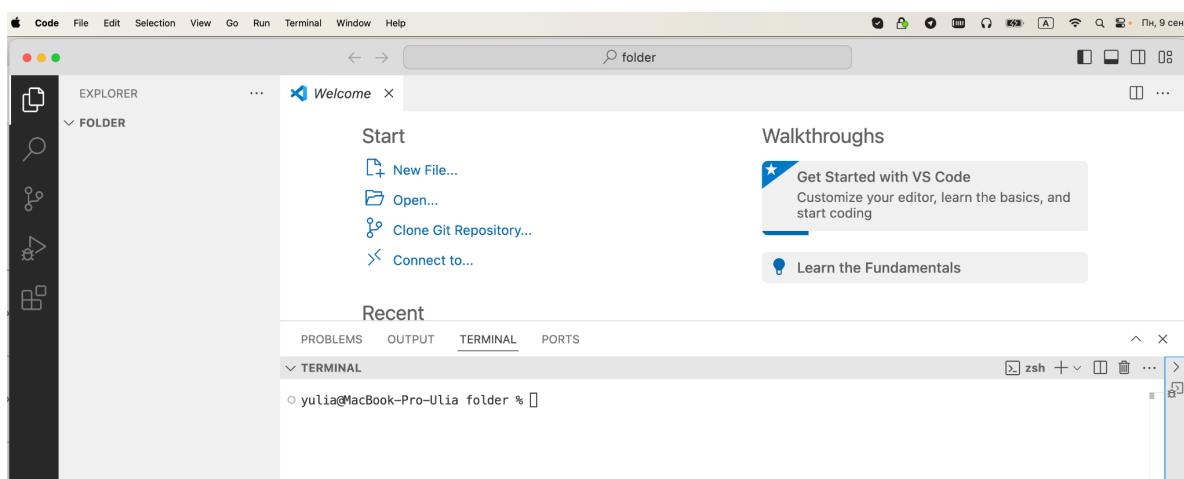
команда	объяснение
<i>mkdir folder</i>	создание папки с именем folder
<i>cd folder</i>	вход в папку <i>folder</i>

<code>pwd</code>	проверка где находимся, убеждаемся, что действительно в папке <i>folder</i>
------------------	---

Для навигации по файловой системе из командной строки достаточно знать несколько команд. Базовыми командами являются `cd` – перейти, `pwd` – проверить где находимся, `ls` – посмотреть содержимое папки в которой находимся.

команда	объяснение
<code>cd /полный/путь/к/папке</code>	переход в любую папку по полному пути
<code>cd имя</code>	переход в подкаталог, то есть папка должна быть там где мы находимся
<code>cd ..</code>	переход на один уровень выше
<code>ls</code>	просмотр содержимого папки в которой находимся
<code>ls -a</code>	просмотр содержимого со скрытыми папками
<code>pwd</code>	проверка где находимся

В качестве среды разработки рекомендуем VSCode. Откроем папку проекта в VSCode. Для выполнения команд в терминале можно остаться в консольном приложении, а можно открыть терминал в VSCode.



В папке проекта *folder* выполним команду

```
$ npm init
```

Аббревиатура *npm* – это *node package manager*, то есть менеджер управления модулями Node.js (готовыми, написанными на JavaScript пакетами). *npm* автоматически устанавливается вместе с Node.js.

```
% npm init
```

This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (folder)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to /Users/yulia/folder/package.json:

```
{
  "name": "folder",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": ""
}
```

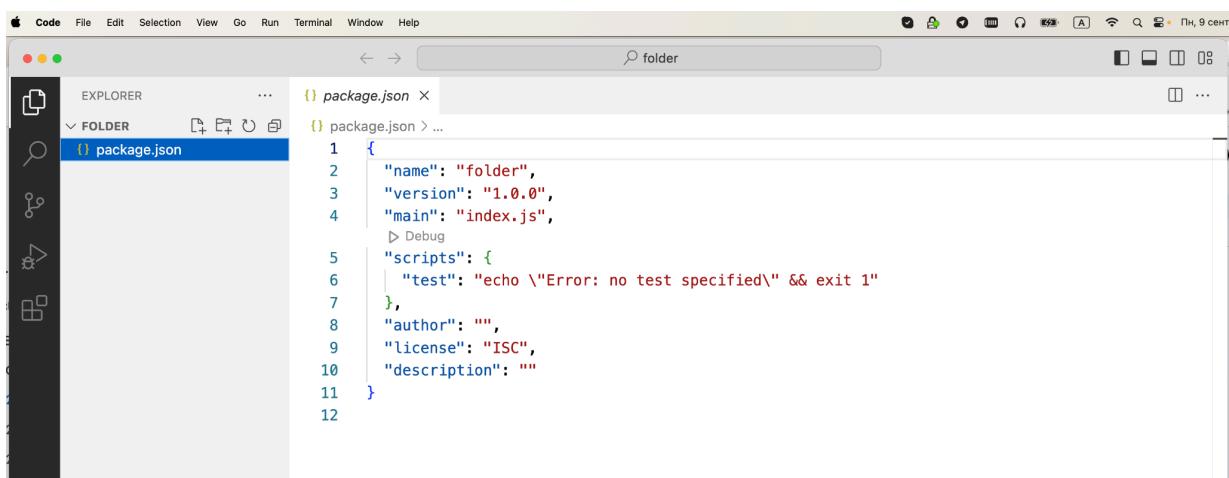
Is this OK? (yes) yes

Команда *npm init* создала в папке *folder* файл *package.json*.

Вернемся в среду разработки для проверки факта появления файла *package.json* в папке проекта и проверки его содержимого.

Листинг файла package.json

```
{  
  "name": "folder",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```



Формат файла – это *JSON (JavaScript Object Notation)* или представление объектов JavaScript) формат. Что такое *JSON*? *JSON (JavaScript Object Notation)* – это способ хранения данных, которые организованы по следующим правилам:

- данные состоят из пар имя/значение;
- пары заключаются либо в фигурные скобки {}, либо в квадратные [] и разделяются запятыми. В случае фигурных скобок данные называют объектом, в случае квадратных – массивом;
- пара имя/значение состоит из имени, заключенного в кавычки, за которым следует : и значение
- значение может быть числом (целым или с точкой), строкой, логическим значением (true или false), массивом, объектом или значением null

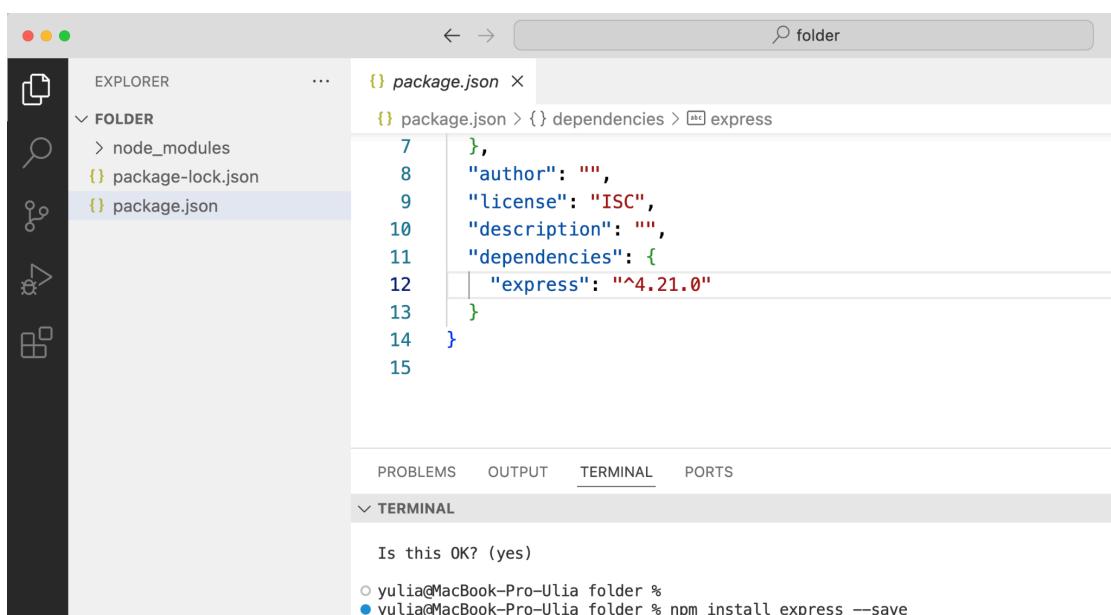
В консоле выполним следующую команду.

```
$ npm install express --save
```

В директории *folder* появляется папка *node_modules*. В эту папку *npm* будет складывать все Node.js модули, которые понадобятся для разработки проекта. *--save* добавляет имя и версию установленного модуля в *package.json* файл.

Листинг файла package.json

```
{
  "name": "folder",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.21.0"
  }
}
```



Создадим в папке *folder* файл *app.js*. Для создания простейшего проекта в файле *app.js* достаточно написать 6 строчек кода.

Листинг файла app.js

```
var express = require("express")
var app = express()

app.get("/",function(req,res){
    res.send("Серверная информация для браузера")
})

app.listen("3000")
```

В этих шести строчках:

1-ая строчка – подключаем модуль *express*,
2-ая строчка - инициализируем экземпляр *express*,
3-5-ые строчки - прописываем обработку рутового маршрута браузера */*,
6-ая строчка - командой *listen()* запускаем Node.js сервер, *3000* – порт доступа из браузера к запущенному серверу.

Перейдем к запуску проекта.

```
$ node app.js
```

Откроем теперь браузер. На странице <http://localhost:3000/> проверяем результат. Текст, отправленный с сервера должен появиться в браузере.

Для остановки Node.js сервера, в консольном приложении выполним инструкцию из ответа npm на команду *npm init*.

Press ^C at any time to quit.

Нажимаем ^C (ctrl+c).

Таким образом решена следующая задача: На странице браузера <http://localhost:3000/> получена информация, переданная с Node.js сервера.

Простейший проект создан. Рассмотрим в его рамках несколько возможностей express.

2.2. ОБРАБОТКА POST ЗАПРОСОВ

Остановимся подробнее на обработке post запросов.

Обмен данными между браузером и сервером происходит с помощью HTTP (HyperText Transfer Protocol) запросов, самые базовые методы это get и post. Набранный в адресной строке браузера url инициирует get запрос. Поэтому для обработки запроса `http://localhost:3000/` в коде сервера использовался метод `app.get`.

Сделаем следующую модификацию этого метода в коде сервера.

В листинге файла app.js

```
app.get('/',function(req,res){
  var form = '<!doctype html>'+
    '<html lang="ru">'+
    '<head>'+
    '<meta charset="UTF-8">'+
    '<title>Форма</title>'+
    '</head>'+
    '<body>'+
    '<h1>Форма для отправки данных на сервер</h1>'+
    '<form action="" method="post">'+
    '<textarea name="" id="" cols="30" rows="10"></textarea><br/>'+
    '<input type="submit" value="Отправить данные на сервер"/>'+
    '</form>'+
    '</body>'+
    '</html>'
  res.send(form)
})
```

Прежде чем запустить сервер добавим в функцию `listen` так называемую `callback` функцию – функцию, которая отрабатывает после завершения работы вызванной функции. В случае вызова express функции `listen`, `callback` функция отработает после завершения работы функции `listen`, т.е. после запуска сервера.

В листинге файла app.js

```
app.listen("3000", function(){
  console.log( "Сервер работает и слушает порт: 3000")
})
```

JavaScript функция `console.log()` делает запись в Console и используется для контроля процесса разработки.

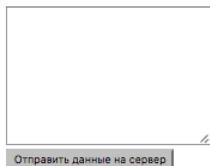
Запустим сервер

```
$ node app.js
```

Сервер работает и слушает порт: 3000

и перейдем в браузер. По адресу `http://localhost:3000/`, на странице должна отобразиться форма

Форма для отправки данных на сервер



A screenshot of a web browser window. Inside, there is a simple form consisting of a single-line text input field and a blue rectangular button below it. The button has the text "Отправить данные на сервер" (Send data to server) in white.

Подготовим обработку запроса `post`. Для этого:

Первое: добавим маршрутизатор для запроса с адресом `"/submit_result"`.

В листинге файла `app.js`

```
app.post("/submit_result", function(req, res){  
  res.send("Вы нажали на кнопку с типом submit");  
})
```

Место в структуре файла `app.js` (между `app.get` и `app.listen`)

```
app.get('/',function(req,res){  
  var form = '<!doctype html>'+  
    '<html lang="ru">'+  
    '<head>'+  
    '<meta charset="UTF-8">'+  
    '<title>Форма</title>'+  
    '</head>'+  
    '<body>'+  
    '<h1>Форма для отправки данных на сервер</h1>'+  
    '<form action="/submit_result" method="post">'+  
    '<textarea name="text"></textarea><br/><br/>'+  
    '<input type="submit" value="Отправить данные на сервер"/>'+  
    '</form>'+  
    '</body>'+
```

```

'</html>'
res.send(form)
})

app.post("/submit_result", function(req, res){
  res.send("Вы нажали на кнопку с типом submit");
})

app.listen("3000",function(){
  console.log( "Сервер работает и слушает порт: 3000")
})

```

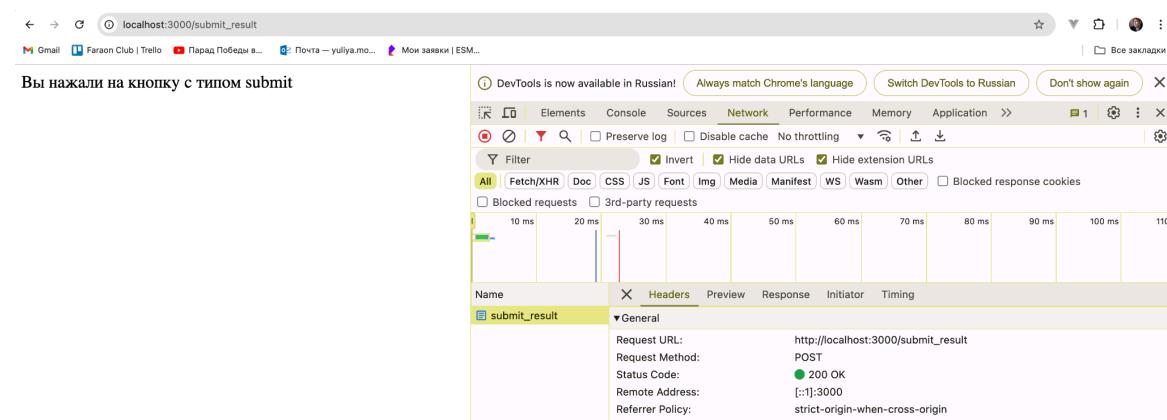
Второе: добавим в форму `action="/submit_result"` – адрес отправки post запроса.

Перейдем к запуску проекта.

```
$ node app.js
```

В окне браузера кликнем правой кнопкой мыши, выберем `Inspect`, откроем вкладку `Network` и кликнем на кнопку формы. Во вкладке `Network` появится строка, содержащая адрес `submit_result` и в браузере откроется страница `http://localhost:3000/submit_result` с текстом «Вы нажали кнопку с типом submit». Если открыть `submit_request` запрос, то во вкладке `Headers` можно убедиться, что к серверу обратился запрос типа `post` и во вкладке `Response` проверить что ответ, совпадает с текстом на странице `http://localhost:3000/submit_result`.

Формы предназначены для передачи данных на сервер.



Чтобы данные из формы были включены в серверный запрос для элементов формы определяется атрибут name. Добавим элементу формы textarea атрибут name="text".

Остановим сервер: ^C.

Запустим сервер: \$node app.js

Перейдем в браузер.

Теперь во вкладке Headers содержатся данные формы.

The screenshot shows the Chrome DevTools Network tab. A single request is listed under the 'All' tab, labeled 'submit_result'. The 'Headers' section is expanded, showing the key 'text' with the value 'Ky-ky'.

Для извлечения данных формы из запроса на сервере установим и подключим модуль body-parser.

\$ npm install body-parser --save

The screenshot shows the VS Code interface with the package.json file open. The 'dependencies' section includes 'body-parser': '^1.20.3' and 'express': '^4.21.0'.

```
7  },
8  "author": "",
9  "license": "ISC",
10 "description": "",
11 "dependencies": [
12   "body-parser": "^1.20.3",
13   "express": "^4.21.0"
14 ]
15 }
16 }
```

Синтаксис подключения модуля body-parser:

```
var bodyParser = require('body-parser');

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

Место в структуре файла app.js (между app.get и app.listen)

```
var express = require("express")
var bodyParser = require("body-parser")

var app = express()
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))
```

```
app.get('/',function(req,res){
  var form = '<!doctype html>'+
  ...
```

Синтаксис извлечения данных формы:

```
app.post("/submit_result",function(req,res){
  console.log(req.body)
  res.send("Вы нажали на кнопку с типом submit")
})
```

Остановим сервер: ^C.

Запустим сервер: \$node app.js

В форму наберем текст «Текст», нажмем кнопку формы и вернемся в консольное приложение.

```
^C
$ node app.js
Сервер работает и слушает порт: 3000
{ text: 'Текст' }
```

Сделаем заключительную правку и вернем на страницу ответа введенную информацию.

```
app.post("/submit_result",function(req,res){
  console.log(req.body)
  var post_text = req.body.text? "Вы отправили на сервер текст: " + req.body.text :
  "Вы отправили на сервер пустую строку"
```

```
    res.send(post_text)
})
```

Остановим сервер: ^C.

Запустим сервер: \$node app.js

Листинг файла app.js

```
var express = require("express")
var bodyParser = require("body-parser")

var app = express()
app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

app.get('/',function(req,res){
  var form = '<!doctype html>'+
    '<html lang="ru">'+
    '<head>'+
    '<meta charset="UTF-8">'+
    '<title>Форма</title>'+
    '</head>'+
    '<body>'+
    '<h1>Форма для отправки данных на сервер</h1>'+
    '<form action="/submit_result" method="post">'+
    '<textarea name="text"></textarea><br/><br/>'+
    '<input type="submit" value="Отправить данные на сервер"/>'+
    '</form>'+
    '</body>'+
    '</html>'
  res.send(form)
})

app.post("/submit_result",function(req,res){
  console.log(req.body)
  var post_text = req.body.text? "Вы отправили на сервер текст: " + req.body.text :
  "Вы отправили на сервер пустую строку"
  res.send(post_text)
})

app.listen("3000",function(){
  console.log( "Сервер работает и слушает порт: 3000")
})
```

2.3. СТАРТОВЫЙ ПРОЕКТ EXPRESS

Рассмотрим использование стартового проекта express для создания приложения.

Одной из приятных возможностей express является наличие стартового проекта, то есть проекта, который настроен для быстрого начала разработки и позволяет сосредоточится на базовых основах программирования.

Для установки и запуск стартового проекта откроем консольное приложение и выполним 2 команды:

```
$ npm install express-generator -g  
$ express -e puh
```

Первая команда – это просто npm установка Node.js модуля express-generator, параметр -g означает глобальную установку (от слова global). Следующий шаг – это собственно создание стартового клиент-сервер проекта на Node.js, puh – название папки в которую будет развернут проект, параметр -e это установка стартового пакета с шаблонизатором ejs. Если выполнить команду без параметра -e то установится шаблонизатор jade. В настоящем пособии для разработки учебного проекта выбран шаблонизатор ejs, так как ejs очень похож на HTML и требует минимального времени для изучения.

```
create : puh  
create : puh/package.json  
create : puh/app.js  
create : puh/public  
create : puh/public/javascripts  
create : puh/public/images  
create : puh/public/stylesheets  
create : puh/public/stylesheets/style.css  
create : puh/routes  
create : puh/routes/index.js  
create : puh/routes/users.js  
create : puh/views  
create : puh/views/index.jade  
create : puh/views/layout.jade  
create : puh/views/error.jade  
create : puh/bin  
create : puh/bin/www
```

```
install dependencies:  
$ cd puh && npm install
```

```
run the app:  
$ DEBUG=puh:* npm start
```

Комментарии, которые пишет консоль достойны прочтения и следования им. Перед тем как перейти к разработке следуя этим инструкциям, откроем папку `ruh` в IDE для удобного отслеживания происходящих изменений. Как и комментировал консоль – в папке `ruh` созданы файлы стартового пакета. В консольном приложении выполним команды и выполним инструкции консоли.

команда	объяснение
<code>cd ruh</code>	вход в папку <code>ruh</code>
<code>pwd</code>	проверка где находимся, убеждаемся, что действительно в папке <code>ruh</code>
<code>npm install</code>	установка модулей из списка <code>"dependencies": { "body-parser": "~1.15.1", "cookie-parser": "~1.4.3", "debug": "~2.2.0", "express": "~4.13.4", "jade": "~1.11.0", "morgan": "~1.7.0", "serve-favicon": "~2.3.0" } }</code> файла <code>package.json</code>
<code>npm start</code>	запуск скелета

Перейдем в браузер и в адресной строке введем `http://localhost:3000` и увидим страницу приветствия.

2.4. Анализ кода стартового проекта

Откроем файл `app.js`. Исследуем значение `app.get('env')`. Из кода видно, что переменная `'env'` принимает два значения: `development` и `production`. `'env'` это переменная среды разработки. Управляется эта переменная командами:

команда	объяснение
<code>export NODE_ENV=production</code>	значение <code>'env'</code> устанавливается равным <code>production</code>

export NODE_ENV=development	значение 'env' устанавливается равным development
-----------------------------	---

Рассмотрим как работает debug модуль. Для использования модуля debug нужно: подключить, выполнить debug, запустить приложение в debug режиме.

подключение:	<code>var debug = require('debug')('puh:server')</code>
выполнение:	<code>debug("Проверка работы debug модуля")</code>
старт приложения:	<code>\$ DEBUG=puh:* npm start</code>
отладка:	<code>puh:server</code> проверка работы debug модуля <code>+0ms</code> <code>puh:server</code> listening on port 3000 <code>+51ms</code>
старт приложения: Замечание от Руслана Фазилова для Windows	<code>\$env:DEBUG='puh: *'; npm start</code>

Рассмотрим механизм отображения страницы приветствия по запросу `http://localhost:3000`. В файле `routes/index.js` код

```
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

управляет обработкой адреса. “index” – это название шаблона, который отобразиться в браузере, title - это переменная, значение которой «Express» передается в шаблон. Шаблон располагается в файле `views/index.ejs`.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Как уже упоминалось – синтаксис шаблонизатора ejs очень прост в изучении. При выводе данных в браузере надо иметь в виду существование спецсимволов: двойная кавычка, амперсанд, апостроф, знак меньше, знак больше. Экранированный вывод – это вывод, когда спецсимволы выводятся как обычные символы. Соответственно неэкранированный – это вывод HTML.

<%= title %> - экранированный вывод данных
<%- title %> - неэкранированный вывод данных

Добавим в значение переменной title спецсимволы:

```
/* GET home page. */  
router.get('/', function(req, res, next) {  
  res.render('index', { title: '<h1>Express</h1>' });  
});
```

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Откроем браузер:



<h1>Express</h1>

Welcome to <h1>Express</h1>

Итак, <%= title %> вывод – это вывод, когда спецсимволы выводятся как обычные символы, то есть экранированный вывод. Изменим теперь <%= title %> вывод на <%- title %> – неэкранированный вывод (файл views/index.ejs). Теперь в браузере спецсимволы распознаются как спецсимволы и теги <h1> интерпретируются как HTML.

Сведем правила синтаксиса ejs в таблицу.

ejs синтаксис	объяснение
<% код JavaScript %>	код JavaScript для формирования шаблона
<%= переменная %>	экранированный вывод

<%- переменная %>	неэкранированный вывод
-%>	закрывающий тег предотвращающий новую строку шаблона
<%_ _%>	удаление пробелов

2.5. Выполнение кода JAVASCRIPT

Напомним, что код JavaScript выполняется двумя способами: в браузере или Node.js.

Выполнение JavaScript в браузере

Для выполнения в браузере можно создать файл с расширением html и в теге script писать JavaScript. Файл можно создавать в любом месте файловой системы. Например, создадим файл file.html:

Листинг файла file.html

```
<!doctype html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>JavaScript</title>
</head>
<body>
<script>
  //Код JavaScript
</script>
</body>
</html>
```

Выполнить код – это открыть html файл в браузере. В окне браузера кликнуть правой кнопкой, выбрать Inspect и открыть вкладку Console. Код JavaScript можно писать в теге script, а можно подключить js файл и писать код во внешнем файле. Для этого создадим папку folder (папку можно создать в любом месте файловой системы). В папке folder создадим 2 файла: file.html, file.js.

Для выполнения кода нужно тоже открыть html файл в браузере. И в окне браузера кликнуть правой кнопкой мыши и выбрать Inspect и открыть вкладку Console.

Выполнение JavaScript с Node.js

Для выполнения JavaScript с использованием Node.js переходим в папку folder и набираем команду:

```
$ node file.js
```

Выполнить команду это нажать клавишу enter.

3. MongoDB

3.1. Консоль MONGO

Как и для Node.js, установка MongoDB зависит от операционной системы и как для любого процесса инсталляции, пошаговая, подробная инструкция ищется, по ключевым словам (например, «установка MongoDB»), в интернете.

В консольном приложении выполним команду:

```
% mongod -version
db version v5.0.7
Build Info: {
    "version": "5.0.7",
    "gitVersion": "b977129dc70eed766cbee7e412d901ee213acbda",
    "modules": [],
    "allocator": "system",
    "environment": {
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

Рассмотрим базовые команды

команда	объяснение
brew services start mongodb-community	запуск базы MongoDB
brew services stop mongodb-community	остановка MongoDB
mongosh	запуск консоли MongoDB
cls	очистка экрана консоли
show dbs	список баз данных
use name	переключение на базу name
db.help()	список доступных функций
db.getCollectionNames()	список коллекций базы
db.getName()	проверка в какой базе находимся
db.collection_name.insert()	добавление данных в коллекцию collection_name
.exit	выход из консоли mongosh

Войдем в консоль MongoDB (mongosh) и выполним команду show dbs

```
$ mongo  
> show dbs  
test 0.000GB
```

В MongoDB ровно одна база с именем test. Следующим шагом выполним команды use learn и db.unicorns.insert:

```
> use learn  
switched to db learn  
> db.unicorns.insert({name: 'Aurora', color: 'white', weight: 450})  
WriteResult({ "nInserted" : 1 })  
> db.getCollectionNames()  
[ "unicorns" ]  
> db.unicorns.count()  
1
```

И еще раз show dbs:

```
> show dbs
```

```
test 0.000GB  
learn 0.000GB
```

Итак, к списку баз данных добавилась новая база данных learn. Таким образом для создания базы данных из консоли достаточно переключиться и добавить данные.

Рассмотрим терминологию и структуру хранения данных в MongoDB.

термины	структура
база	база состоит из коллекций
коллекция	коллекция состоит из документов
документ	документ состоит из пар поле/значение поля

Для данных unicorns:

```
learn – база  
unicorns - коллекция  
{name: 'Aurora', color: 'white', weight: 450} – документ  
name, color, weight – поля
```

Удалим базу данных learn и проверим, что база данных удалена.

```
> show dbs  
test 0.000GB  
learn 0.000GB  
> use learn  
switched to db learn  
> db.dropDatabase()  
{ "dropped" : "learn", "ok" : 1 }  
> show dbs  
test 0.000GB
```

Подготовим базу данных для выполнения упражнений:

```
> use learn  
switched to db learn
```

Скопируем и вставим в консоль добавление документов в коллекцию unicorns.

```
db.unicorns.insert({name: 'Horny', birthday: new Date(1992,2,13,7,47), loves: ['carrot','papaya'], weight: 600, color: 'black', vampires: 63});  
db.unicorns.insert({name: 'Aurora', birthday: new Date(1991, 0, 24, 13, 0), loves: ['carrot', 'grape'], weight: 450, color: 'white', vampires: 43});  
db.unicorns.insert({name: 'Unicrom', birthday: new Date(1973, 1, 9, 22, 10), loves: ['energon', 'redbull'], weight: 984, color: 'black', vampires: 182});  
db.unicorns.insert({name: 'Roodles', birthday: new Date(1979, 7, 18, 18, 44), loves: ['apple'], weight: 575, color: 'black', vampires: 99});  
db.unicorns.insert({name: 'Solnara', birthday: new Date(1985, 6, 4, 2, 1), loves:[apple', 'carrot', 'chocolate'], weight:550, color:'white', vampires:80});  
db.unicorns.insert({name:'Ayna', birthday: new Date(1998, 2, 7, 8, 30), loves: ['strawberry', 'lemon'], weight: 733, color: 'white', vampires: 40});  
db.unicorns.insert({name:'Kenny', birthday: new Date(1997, 6, 1, 10, 42), loves: ['grape', 'lemon'], weight: 690, color: 'black', vampires: 39});  
db.unicorns.insert({name: 'Raleigh', birthday: new Date(2005, 4, 3, 0, 57), loves: ['apple', 'sugar'], weight: 421, color: 'black', vampires: 2});  
db.unicorns.insert({name: 'Leia', birthday: new Date(2001, 9, 8, 14, 53), loves: ['apple', 'watermelon'], weight: 601, color: 'white', vampires: 33});  
db.unicorns.insert({name: 'Pilot', birthday: new Date(1997, 2, 1, 5, 3), loves: ['apple', 'watermelon'], weight: 650, color: 'black', vampires: 54});  
db.unicorns.insert({name: 'Nimue', birthday: new Date(1999, 11, 20, 16, 15), loves: ['grape', 'carrot'], weight: 540, color: 'white'});  
db.unicorns.insert({name: 'Dunx', birthday: new Date(1976, 6, 18, 18, 18), loves: ['grape', 'watermelon'], weight: 704, color: 'black', vampires: 165});
```

Убедимся, что создана база данных learn, что создана коллекция unicorns и все 12 документов добавлены.

```
> show dbs  
learn 0.000GB  
test 0.000GB  
> db.getName()  
learn  
> db.getCollectionNames()  
[ "unicorns" ]  
> db.unicorns.count()  
12
```

Итак, мы подняли MongoDB, изучили структуру, создали базу learn, в базе learn коллекцию unicorns и добавили в эту коллекцию 12 документов. Для добавления документов в коллекцию была использована команда insert.

3.2. Команда FIND

Изучим поиск с условиями: и, или(\$or), не из списка (\$nor), меньше(\$lt), меньше или равно(\$lte), больше(\$gt), больше или равно(\$gte), не равно(\$ne), существует(\$exists).

1	Найти черных единорогов, которые весят больше 700
2	Найти не черных единорогов вес которых не меньше 701
3	Найти единорогов без статистики убитых вампиров
4	Найти белых единорогов которые весят меньше 500
5	Найти сколько единорогов не весят 600
6	Найти черных единорогов которые любят арбузы или весят больше 900
7	Найти сколько единорогов любят яблоки или морковку
8	Найти сколько единорогов не любят яблоки и не любят морковку
9	Найти единорогов, которые любят виноград или морковку
10	Найти единорогов которые любят яблоки или убили вампиров не меньше 63

Решение:

1. `db.unicorns.find({color:"black",weight:{$gt: 700}}).`

Синтаксис запроса – это фигурные скобки {} и условия, которые должны выполняться одновременно через запятую, реализация запроса с условием «и»: единорог должен быть черным И(запятая) вес его должен быть больше 700. Условие для поля weight (weight > 700) должно быть в формате :{\$gt: 700}.

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({color:"black",weight:{$gt: 700}})
{ "_id" : ObjectId("598dab792ac4bb803a663664"), "name" : "Unicrom", "birthday" :
ISODate("1973-02-09T19:10:00Z"), "loves" : [ "energon", "redbull" ], "weight" : 984, "color"
: "black", "vampires" : 182 }
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" :
ISODate("1976-07-18T15:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704,
"color" : "black", "vampires" : 165 }
```

При добавлении документа в базу, MongoDB добавляет к документу поле `_id` с уникальным значением.

2. `db.unicorns.find({color:{$ne:"black"},weight:{$gte:701}})`

Используем синтаксис \$ne (не равно) и условие \$gte (больше или равно)

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({color:{$ne:"black"},weight:{$gte:701}})  
{ "_id" : ObjectId("598dab792ac4bb803a663667"), "name" : "Ayna", "birthday" :  
ISODate("1998-03-07T06:30:00Z"), "loves" : [ "strawberry", "lemon" ], "weight" : 733,  
"color" : "white", "vampires" : 40 }
```

Запрос вернул одного единорога.

3. **db.unicorns.find({vampires:{\$exists:false}})**

Запрос на документы, в которых отсутствует поле vampires – {\$exists:false}. Одно из ключевых свойств MongoDB – возможность хранить документы с разными наборами полей в одной коллекции. Так в коллекции unicorns есть документы в которых задано поле vampires и есть документы в которых это поле не задано. Запрос должен найти документы без поля vampires.

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({vampires:{$exists:false}})  
{ "_id" : ObjectId("598dab792ac4bb803a66366c"), "name" : "Nimue", "birthday" :  
ISODate("1999-12-20T14:15:00Z"), "loves" : [ "grape", "carrot" ], "weight" : 540, "color" :  
"white" }
```

4. **db.unicorns.find({color:"white",weight:{\$lt:500}})**

Для условия меньше или равно используем синтаксис {\$lt:500}.

Выполним запрос в консоли MongoDB:

```
db.unicorns.find({color:"white",weight:{$lt:500}})  
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" :  
ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "color" :  
"white", "vampires" : 43 }
```

5. **db.unicorns.find({weight:{\$ne:600}}).count()**

Применение условия \$ne (не равно) уже знакомо. Новым синтаксисом является команда вычисления количества count().

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({weight:{$ne:600}}).count()
```

11

6 **db.unicorns.find({color:"black"},{\$or:[{loves:"watermelon"},{weight:{\$gt:900}}]})**

Синтаксис условия \$or (или) это условие \$or, двоеточие, квадратные скобки, в квадратных скобках через запятую условия. Еще одним важным правилом запросов в MongoDB является обращение к элементам массивов. Поле love в документе массив, условие – {loves:"watermelon"} это поиск по элементу массива. Запрос будет искать единорогов у которых в списке loves есть арбуз.

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({color:"black",$or:[{loves:"watermelon"},{weight:{$gt:900}}]})  
{ "_id" : ObjectId("598dab792ac4bb803a663664"), "name" : "Unicrom", "birthday" :  
ISODate("1973-02-09T19:10:00Z"), "loves" : [ "energon", "redbull" ], "weight" : 984, "color"  
: "black", "vampires" : 182 }  
{ "_id" : ObjectId("598dab792ac4bb803a66366b"), "name" : "Pilot", "birthday" :  
ISODate("1997-03-01T02:03:00Z"), "loves" : [ "apple", "watermelon" ], "weight" : 650,  
"color" : "black", "vampires" : 54 }  
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" :  
ISODate("1976-07-18T15:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704,  
"color" : "black", "vampires" : 165 }  
>
```

Получили три единорога. Единорог Unicorn был найден потому что черный и весит больше 900 (арбузы не любит). Единорог Pilot был найден потому что он черный и потому что любит арбузы (вес у него меньше 900). Единорог Dunx, найден как и Pilot, потому что черный и потому что любит арбузы, хотя вес его меньше 900.

7 db.unicorns.find({\$or:[{loves:'apple'}, {loves:'carrot'}]}).count()

Повторяем синтаксис запроса \$or(или) и обращение к элементам массива.

Выполним запрос в консоли MongoDB:

```
> db.unicorns.find({$or:[{loves:'apple'}, {loves:'carrot'}]}).count()  
8
```

Найдено восемь единорогов.

8 db.unicorns.find({\$nor:[{loves:'apple'}, {loves:'carrot'}]})

Изучим \$nor условие.

Выполним запрос в консоли MongoDB:

```

> db.unicorns.find({$nor:[{loves:'apple'}, {loves:"carrot"}]})

{ "_id" : ObjectId("598dab792ac4bb803a663664"), "name" : "Unicrom", "birthday" :
ISODate("1973-02-09T19:10:00Z"), "loves" : [ "energon", "redbull" ], "weight" : 984, "color"
: "black", "vampires" : 182 }
{ "_id" : ObjectId("598dab792ac4bb803a663667"), "name" : "Ayna", "birthday" :
ISODate("1998-03-07T06:30:00Z"), "loves" : [ "strawberry", "lemon" ], "weight" : 733,
"color" : "white", "vampires" : 40 }
{ "_id" : ObjectId("598dab792ac4bb803a663668"), "name" : "Kenny", "birthday" :
ISODate("1997-07-01T07:42:00Z"), "loves" : [ "grape", "lemon" ], "weight" : 690, "color" :
"black", "vampires" : 39 }
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" :
ISODate("1976-07-18T15:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704,
"color" : "black", "vampires" : 165 }
>

```

Найдено 4 единорога. И они любят все что угодно кроме яблок и морковки.

9. db.unicorns.find({\$or:[{loves:'grape'}, {loves:"lemon"}]})

Повторяем синтаксис запроса \$or(или) и обращение к элементам массива.

Выполним запрос в консоли MongoDB:

```

> db.unicorns.find({$or:[{loves:'grape'}, {loves:"lemon"}]})

{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" :
ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "color" :
"white", "vampires" : 43 }
{ "_id" : ObjectId("598dab792ac4bb803a663667"), "name" : "Ayna", "birthday" :
ISODate("1998-03-07T06:30:00Z"), "loves" : [ "strawberry", "lemon" ], "weight" : 733,
"color" : "white", "vampires" : 40 }
{ "_id" : ObjectId("598dab792ac4bb803a663668"), "name" : "Kenny", "birthday" :
ISODate("1997-07-01T07:42:00Z"), "loves" : [ "grape", "lemon" ], "weight" : 690, "color" :
"black", "vampires" : 39 }
{ "_id" : ObjectId("598dab792ac4bb803a66366c"), "name" : "Nimue", "birthday" :
ISODate("1999-12-20T14:15:00Z"), "loves" : [ "grape", "carrot" ], "weight" : 540, "color" :
"white" }
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" :
ISODate("1976-07-18T15:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704,
"color" : "black", "vampires" : 165 }
>

```

Это единороги: Aurora, Ayna, Kenny, Nimue, Dunx.

10 db.unicorns.find({\$or:[{loves:"apple"}, {vampires:\$gte:63}]})

3.3. Команда UPDATE

Изучим команду update и условия \$set, \$inc, \$push, \$pull, \$mul.

11	Единорог Roodles похудел на 10
12	Единорог Pilot убил еще 2 вампиров
13	Единорог Aurora полюбил сахар
14	Добавить информацию, что Aurora привита (vaccinated: true)
15	Проверить какие единороги привиты
16	Единорог Dunx разлюбил виноград
17	Вес единорога Aurora увеличился в два раза

Решение:

```
11 db.unicorns.update({name:"Roodles"},{$inc:{weight:-10}})
```

Проанализируем синтаксис команды update. У команды update два аргумента. Первый аргумент – это условие поиска, второй аргумент – это описание изменения, которое нужно сделать. Аргументы разделены запятой. Условие поиска – единорог по имени Roodles. Условие \$inc – изменяет значение поля weight на величину –10.

Сначала выполним в консоли MongoDB поиск единорога с именем Roodles: db.unicorns.find({name:"Roodles"})

```
> db.unicorns.find({name:"Roodles"})
{ "_id" : ObjectId("598dab792ac4bb803a663665"), "name" : "Roodles", "birthday" :
ISODate("1979-08-18T15:44:00Z"), "loves" : [ "apple" ], "weight" : 575, "color" : "black",
"vampires" : 99 }
```

Вес Roodles – 575.

Теперь сделаем update поля weight и снова извлечем сведения о единороге с именем Roodle из базы.

```
> db.unicorns.update({name:"Roodles"},{$inc:{weight:-10}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.unicorns.find({name:"Roodles"})
{ "_id" : ObjectId("598dab792ac4bb803a663665"), "name" : "Roodles", "birthday" :
ISODate("1979-08-18T15:44:00Z"), "loves" : [ "apple" ], "weight" : 565, "color" : "black",
"vampires" : 99 }
>
```

Вес Roodles стал 565, то есть уменьшился на 10.

12 db.unicorns.update({name:"Pilot"},{\$inc:{vampires:2}})

В этом упражнении отрабатывается уже рассмотренное выше условие \$inc.

13 db.unicorns.update({name:"Aurora"},{\$push:{loves:"sugar"}})

Условие запроса \$push добавит к списку сахар.

Сначала убедимся, что единорог с именем Aurora не любит сахар:

```
db.unicorns.find({name:"Aurora"})
```

```
> db.unicorns.find({name:"Aurora"})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" :
ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape" ], "weight" : 450, "color" :
"white", "vampires" : 43 }
```

Действительно Aurora любит морковку и виноград.

Теперь выполним update и снова запросим информацию об единороге Aurora.

```
> db.unicorns.update({name:"Aurora"},{$push:{loves:"sugar"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.unicorns.find({name:"Aurora"})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" :
ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 450,
"color" : "white", "vampires" : 43 }
```

14 db.unicorns.update({name:"Aurora"},{\$set:{vaccinated:true}})

В этом упражнении используется условие \$set – установить, но особенно интересно, что можно устанавливать значение поля, которого в документе нет.

```
> db.unicorns.find({name:"Aurora"})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" :
ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 450,
"color" : "white", "vampires" : 43 }
```

```
> db.unicorns.update({name:"Aurora"},{$set:{vaccinated:true}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.unicorns.find({name:"Aurora"})
```

```
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" : ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 450, "color" : "white", "vampires" : 43, "vaccinated" : true }
```

15 db.unicorns.find({vaccinated:true})

```
> db.unicorns.find({vaccinated:true})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" : ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 450, "color" : "white", "vampires" : 43, "vaccinated" : true }
```

16 db.unicorns.update({name:"Dunx"},{\$pull:{loves:"grape"}})

Сначала проверим, что Dunx не любит виноград, затем используем update с условием \$pull и проверим, что полюбил.

```
> db.unicorns.find({name:"Dunx"})
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" : ISODate("1976-07-18T15:18:00Z"), "loves" : [ "grape", "watermelon" ], "weight" : 704, "color" : "black", "vampires" : 165 }
> db.unicorns.update({name:"Dunx"},{$pull:{loves:"grape"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.unicorns.find({name:"Dunx"})
{ "_id" : ObjectId("598dab792ac4bb803a66366d"), "name" : "Dunx", "birthday" : ISODate("1976-07-18T15:18:00Z"), "loves" : [ "watermelon" ], "weight" : 704, "color" : "black", "vampires" : 165 }
```

17 db.unicorns.update({name:"Aurora"},{\$mul:{weight: 2}})

Проверим вес единорога Aurora, выполним команду update и снова проверим вес.

```
> db.unicorns.find({name:"Aurora"})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" : ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 450, "color" : "white", "vampires" : 43, "vaccinated" : true }
```

```
> db.unicorns.update({name:"Aurora"},{$mul:{weight: 2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.unicorns.find({name:"Aurora"})
{ "_id" : ObjectId("598dab792ac4bb803a663663"), "name" : "Aurora", "birthday" : ISODate("1991-01-24T11:00:00Z"), "loves" : [ "carrot", "grape", "sugar" ], "weight" : 900, "color" : "white", "vampires" : 43, "vaccinated" : true }
```

Вес Aurora был 450, стал 900, то есть увеличился в два раза.

3.4. УПРАЖНЕНИЯ ДЛЯ ЗАКРЕПЛЕНИЯ ПРАВИЛ СИНТАКСИСА

Упражнения для закрепления правил синтаксиса.

Исправить ошибки в запросах:

1	db.unicorns.update({name="Roooooodles"},{\$inc:{weight:-10}})
2	db.unicorn.update({name:'Kenny'},{\$set:{weight:603}})
3	db.unicorns.find({gender:"m"},{weight:{\$gt:500}})
4	db.unicorns.find().sort({vampires})
5	db.unicorns.find(name:"Pilot")
6	db.unicorns.update({name:"Pilot", \$pull:{loves:"apple"}})
7	db.unicorns.find({color: "white"},{weight:{\$lt:500}})
8	db.unicorns.update({name:"Pilot"},{ vampires:{ \$inc:2}})
9	db.unicorns.update({name:"Roodles"},\$inc:{weight:-10})
10	db.unicorns.find({color: "white",weight:{\$gt:500}})
11	db.unicorns.find({\$or:[{loves:"apple"},{vampires:{\$ghe:63}}]})

Ответы:

1	db.unicorns.update({name="Roodles"},{\$inc:{weight:-10}})
	db.unicorns.update({name: "Roodles"},{\$inc:{weight:-10}})
2	db.unicorn.update({name:'Kenny'},{\$set:{weight:603}})
	db.unicorns.update({name:'Kenny'},{\$set:{weight:603}})
3	db.unicorns.find({color:"black"},{weight:{\$gt:500}})
	db.unicorns.find({color:"black",weight:{\$gt:500}})
4	db.unicorns.find().sort({vampires})
	db.unicorns.find().sort({vampires:-1})
5	db.unicorns.find(name:"Pilot")
	db.unicorns.find({name:"Pilot"})
6	db.unicorns.update({name:"Pilot", \$pull:{loves:"apple"}})
	db.unicorns.update({name:"Pilot"},{\$pull:{loves:"apple"}})
7	db.unicorns.find({color: "white"},{ weight:{\$lt:500}})
	db.unicorns.find({color: "white", weight:{\$lt:500}})
8	db.unicorns.update({name:"Pilot"},{ vampires:{ \$inc:2}})
	db.unicorns.update({name:"Pilot"},{\$inc:{vampires:2}})
9	db.unicorns.update({name:"Roodles"},\$inc:{weight:-10})
	db.unicorns.update({name:"Roodles"},{\$inc:{weight:-10}})
10	db.unicorns.find({color: "white",weight:{\$gt:500}})
	db.unicorns.find({color: "white",weight:{\$gt:500}})

11	db.unicorns.find({\$or:[{loves:"apple"},{vampires:{\$ghe:63}}]}))
	db.unicorns.find({\$or:[{loves:"apple"},{vampires:{'\$gte:63}}]}))

Продолжить изучение MongoDB с единорогами можно здесь:
[https://docs.mongodb.com/manual/.](https://docs.mongodb.com/manual/)

4. МАРШРУТИЗАТОРЫ И ШАБЛОНЫ СТАРТОВОГО EXPRESS ПРОЕКТА

4.1. СТАРТОВЫЙ ПРОЕКТ С ШАБЛОНIZАТОРОМ EJS

Создадим новый стартовый express проект.

команда	описание
pwd	проверка где находимся, убеждаемся, что не находимся в папке никакого проекта

Выполним команду

```
$ express -e tc2024
```

```
create : tc2024/
create : tc2024/public/
create : tc2024/public/javascripts/
create : tc2024/public/images/
create : tc2024/public/stylesheets/
create : tc2024/public/stylesheets/style.css
create : tc2024/routes/
create : tc2024/routes/index.js
create : tc2024/routes/users.js
create : tc2024/views/
create : tc2024/views/error.jade
create : tc2024/views/index.jade
create : tc2024/views/layout.jade
create : tc2024/app.js
create : tc2024/package.json
create : tc2024/bin/
create : tc2024/bin/www
change directory:
```

```

$ cd tc2024
install dependencies:
$ npm install
run the app:
$ DEBUG=tc2024:* npm start

```

Это создание стартового клиент-сервер проекта на Node.js, *tc2024* – название папки в которую будет развернут проект, параметр *-e* это установка стартового пакета с шаблонизатором *ejs*. Напомним, что если выполнить команду без параметра *-e* то установится шаблонизатор *jade*.

После отчета о создании скелета проекта в консоле прописаны следующие шаги:

команда	объяснение
<i>cd tc2024</i>	вход в папку <i>tc2024</i>
<i>pwd</i>	проверка где находимся, убеждаемся, что действительно в папке <i>tc2024</i>
<i>npm install</i>	<p>установка модулей из списка</p> <pre> "dependencies": { "cookie-parser": "~1.4.4", "debug": "~2.6.9", "ejs": "~2.6.1", "express": "~4.16.1", "http-errors": "~1.6.3", "morgan": "~1.9.1" } </pre> <p>файла <i>package.json</i></p>
<i>npm start</i>	запуск стартового проекта

Перейдем в браузер и в адресной строке введем <http://localhost:3000> и увидим страницу приветствия.

Открыть проект в редакторе (рассматриваем VSCode) можно сразу после команды *express -e tc2024*, можно после запуска проекта.
Открываем папку *tc2024* в VSCode

В папке tc2024 создадим файл .gitignore. Файл .gitignore содержит инструкции по отправке файлов в git репозиторий. Файлы папки node_modules не принято добавлять в git репозиторий так как они добавляются в проект командой npm install.

Листинг файла .gitignore (по запросу gitignore for Express app)

```
# Logs
logs
*.log

# Runtime data
pids
*.pid
*.seed

# Directory for instrumented libs generated by jscoverage/JSCover
lib-cov

# Coverage directory used by tools like istanbul
coverage

# Grunt intermediate storage
(http://gruntjs.com/creating-plugins#storing-task-files)
.grunt

# node-waf configuration
.lock-wscript

# Compiled binary addons (http://nodejs.org/api/addons.html)
build/Release

# Dependency directory
# https://docs.npmjs.com/cli/shrinkwrap#caveats
node_modules

# Debug log from npm
npm-debug.log

.DS_Store

.env
```

После добавления файла .gitignore создаем удаленный репозиторий и делаем push кода проекта в удаленный репозиторий.

4.2. МАРШРУТИЗАТОР

Каждый пункт учебного пособия будем выполнять в отдельной ветке. Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
* main
% git checkout -b tc_4_2
Switched to a new branch 'tc_4_2'
% git branch
  main
* tc_4_2
```

В главном файле приложения app.js найдем строку

```
app.use(express.static(path.join(__dirname, 'public')));
```

Это объявление public папки. Файлы, к которым разрешен доступ из браузера. Остальные файлы проекта защищены из соображений безопасности приложения.

Из структуры папки (в папке public расположены папки images, javascripts, stylesheets) понятно, что для открытого доступа предназначены картинки, клиентская интерактивность и стили. Выберем в интернете картинки, назовем их: korzhik.png, karamelka.jpeg, kompot.png и положим в папку public/images. Эти картинки будут доступны в браузере.

В IDE откроем файл routes/index.js. Под обработкой маршрута «/»

```
/* GET home page. */
router.get('/', function(req, res, next) {
```

```
res.render('index', { title: 'Express' });
});
```

пишем свой код сходной структуры:

```
/* Страница Коржика*/
router.get('/korzhik', function(req, res, next) {
  res.send("<h1>Страница Коржика</h1>")
});
```

В консоле выполним команды:

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

В браузере откроем страницу /korzhik (полный путь <http://localhost:3000/korzhik>). Страница должна отобразить текст «Страница Коржика».

Проделаем это еще 2 раза в файле routes/index.js:

```
/* Страница Карамельки */
router.get('/karamelka', function(req, res, next) {
  res.send("<h1>Страница Карамельки</h1>")
});

/* Страница Компота */
router.get('/kompot', function(req, res, next) {
  res.send("<h1>Страница Компота</h1>")
});
```

И в браузере адреса /korzhik, /karamelka, /kompot отобразят тексты «Страница Коржика», «Страница Карамельки», «Страница Компота» соответственно.

Пушим ветку в удаленный репозиторий

```
% git add .
% git commit -m'Пункт 4_2'
% git push origin tc_4_2
```

4.3. Создание шаблона

Создаем новую ветку для пункта 4.3

```
% git branch
  main
* tc_4_2
% git checkout -b tc_4_3
Switched to a new branch 'tc_4_3'
% git branch
  main
  tc_4_2
* tc_4_3
```

Следующим шагом создадим шаблон для 3-х страниц котов. В VSCode откроем папку views –это папка шаблонов. Создадим в этой папке файл cat.ejs и скопируем в этот файл содержимое views/index.ejs.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Напомним, что в routes/index.js обработка адреса «/» выглядит следующим образом:

```
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

Это значит, что для ответа сервер использует шаблон index и передает в этот шаблон переменную title со значением «Express».

Преобразуем шаблон views/cat.ejs, так чтобы он принимал три переменные title, picture и desc.

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
  <link rel='stylesheet' href='/stylesheets/style.css' />
</head>
<body>
  <h1><%= title %></h1>
  
  <h4>Несколько слов о нашем герое: <%=desc%></h4>
</body>
</html>
```

Вернемся в файл routes/index.js и преобразуем код

```
/* Страница Коржика*/
router.get('/korzhik', function(req, res, next) {
  res.send("<h1>Страница Коржика</h1>")
});
```

по аналогии с

```
/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

получим

```
/* Страница Коржика */
router.get('/korzhik', function(req, res, next) {
  res.render('cat', {
    title: "Коржик",
    picture: "images/korzhik.png",
    desc: "Средний котёнок в семье. Очень любит футбол.  
Любит бегать, прыгать и веселиться. Иногда делает такие  
вещи, что из них приходится выбираться всей семьёй. Одет в  
форму моряка."})
```

```
});  
});
```

Проверим полученный результат. В консоле:

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

В браузере по адресу /vinni (полный путь http://localhost:3000/korzhik) должен отобразится шаблон, заполненный данными о Коржике.

Преобразуем аналогичным образом обработку адресов: /karamelka, /kompot. Получим:

```
/* Страница Карамельки */  
  
router.get('/karamelka', function(req, res, next) {  
  res.render('cat', {  
    title: 'Карамелька',  
    picture: "/images/karamelka.jpeg",  
    desc: "Самый младший член семьи. Коронная фраза – «Я  
знаю, что надо делать!». Носит красный бант и красное  
платье."  
  });  
});  
  
/* Страница Компота */  
  
router.get('/kompot', function(req, res, next) {  
  res.render('cat', {  
    title: 'Компот',  
    picture: "/images/kompong.png",  
    desc: "Старший котёнок в семье. Ходит в школу. Очень  
любит грибы, читает про них энциклопедии. Любит игры на  
логику. Когда очень сильно огорчается, восклицает «Ну
```

вот!» и начинает плакать. Обожает вкусно поесть. Носит зелёный костюм и шапку."

```
});  
});
```

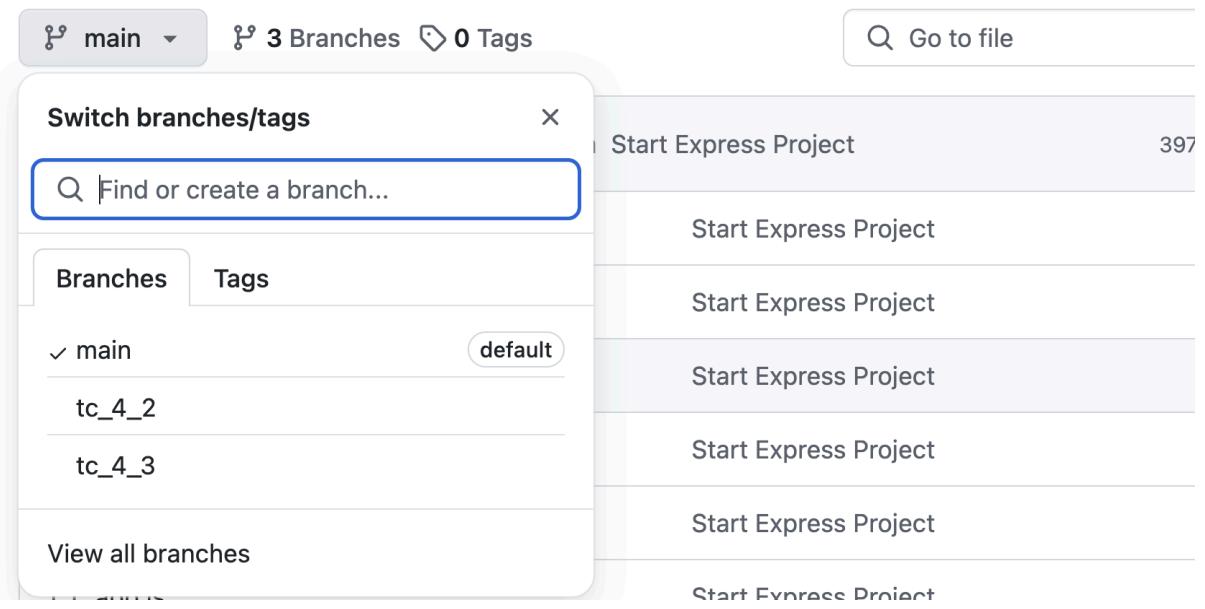
Проверяем:

команда	объяснение
<code>^C</code>	остановка сервера
<code>npm start</code>	запуск сервера

Адреса /korzhik, /karamelka, /kompot отображают шаблон views/cat.js с данными из обработчиков routes/index.js.

Пушим ветку в удаленный репозиторий

```
% git branch  
  main  
  tc_4_2  
* tc_4_3  
% git add .  
% git commit -m'tc_4_3'  
% git branch  
  main  
  tc_4_2  
* tc_4_3  
% git push origin tc_4_3
```



4.4. УСТАНОВКА И ПОДКЛЮЧЕНИЕ ШАБЛОНИЗАТОРА EJS-LOCALS

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  tc_4_2
* tc_4_3
% git checkout -b tc_4_4
Switched to a new branch 'tc_4_4'
% git branch
  main
  tc_4_2
  tc_4_3
* tc_4_4
```

К стандартам разработки web-приложения относится наличие на каждой странице шапки сайта, подвала сайта, боковых панелей. Технически, задача решается созданием шаблона с разметкой для повторяющихся блоков и регионов для уникального контента. Шаблонизатор ejs этого не предусматривает. Поэтому установим и подключим шаблонизатор, который справится с этой задачей.

```
$ npm install ejs-locals --save
```

В файле app.js подключаем шаблонизатор:

```
// view engine setup
app.engine('ejs', require('ejs-locals'));
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

Создадим структуру каждой страницы. Для этого в папке views создадим папку layout. В папке layout, создадим файл page.ejs. скопируем содержимое views/index.ejs и сделаем изменения.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <%- body %>
  </body>
</html>
<body>
```

Неэкранированный вывод <%- body %> обеспечивает вывод уникального контента страницы.

В шаблонах views/index.ejs, views/cat.ejs оставим только содержимое <body></body> шаблонов и в views/index.ejs, views/error.ejs, views/cat.ejs добавим первой строкой каждого файла подключение layout/page.ejs разметки.

```
<% layout('./layout/page.ejs') %>
```

После изменений файлы views/index.ejs, views/error.ejs, views/hero.ejs примут вид:

Листинг файла views/index.ejs

```
<% layout('./layout/page.ejs') %>
```

```
<h1><%= title %></h1>
<p>Welcome to <%= title %></p>
```

Листинг файла views/error.ejs

```
<% layout('./layout/page.ejs') %>
```

```
<h1><%= message %></h1>
<h2><%= error.status %></h2>
<pre><%= error.stack %></pre>
```

Листинг файла *views/hero.ejs*

```
<% layout('./layout/page.ejs') %>
```

```
<h1><%= title %></h1>

<h4>Несколько слов о нашем герое: <%=desc%></h4>
```

Проверяем:

команда	объяснение
<code>^C</code>	остановка сервера
<code>npm start</code>	запуск сервера

Главный результат проверки – поведение клиентской части приложение должно работать без изменений. Это значит, что шаблонизатор подключился и заменят `body` в `<% body %>` из `page.ejs` содержимым шаблонов `views/index.ejs`, `views/error.ejs`, `views/cat.ejs`.

Вызовем отображение шаблона `error.ejs`

Для этого в адресной строке достаточно ввести несуществующий url.
Получим:

```
localhost:3000/yuyuy
Gmail Faraon Club | Trello Парад Победы в... Почта — yuliya.mo... Мои заявки | ESM...

ReferenceError: /Users/yulia/tc2024/views/layout/page.ejs:4
2| <html>
3|   <head>
>> 4|     <title><%= title %></title>
5|     <link rel='stylesheet' href='/stylesheets/style.css' />
6|   </head>
7|   <body>

title is not defined
at eval (eval at exports.compile (/Users/yulia/tc2024/node_modules/ejs-locals/index.js:131:7) at eval (eval at exports.compile (/Users/yulia/tc2024/node_modules/ejs-locals/index.js:131:7) at /Users/yulia/tc2024/node_modules/ejs-locals/node_modules/ejs/lib/ejs.js:284:7 at exports.render (/Users/yulia/tc2024/node_modules/ejs-locals/node_modules/ejs/lib/ejs.js:335:12) at exports.renderFile (/Users/yulia/tc2024/node_modules/ejs-locals/node_modules/ejs/lib/ejs.js:335:12) at module.exports (/Users/yulia/tc2024/node_modules/ejs-locals/index.js:85:7) at /Users/yulia/tc2024/node_modules/ejs-locals/index.js:131:7 at exports.renderFile (/Users/yulia/tc2024/node_modules/ejs-locals/node_modules/ejs/lib/ejs.js:335:12) at module.exports [as engine] (/Users/yulia/tc2024/node_modules/ejs-locals/index.js:85:7) at View.render (/Users/yulia/tc2024/node_modules/express/lib/view.js:135:8)
```

Для исправления ошибки найдем вызов шаблона error.js и добавим переменную title

```
//res.render('error');

res.render('error', {title: 'Three Cats'});
```

Проверяем:

```
localhost:3000/yuyuy
Gmail Faraon Club | Trello Парад Победы в... Почта — yuliya.mo... Мои заявки | ESM...
```

Not Found

404

```
NotFoundError: Not Found
at /Users/yulia/tc2024/app.js:28:8
at Layer.handle [as handle_request] (/Users/yulia/tc2024/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:317:13)
at /Users/yulia/tc2024/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:335:12)
at next (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:275:10)
at /Users/yulia/tc2024/node_modules/express/lib/router/index.js:635:15
at next (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:260:14)
at Function.handle (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:174:3)
at router (/Users/yulia/tc2024/node_modules/express/lib/router/index.js:47:12)
```

Вопрос: Где отображается переменная title?

Пушим ветку в удаленный репозиторий

```
% git branch  
  main  
  ...  
* tc_4_4  
% git add .  
% git commit -m'tc_4_4'  
% git branch  
  main  
  ...  
* tc_4_4  
% git push origin tc_4_4
```

5. Навигация

Стандартное приложение предполагает возможность перехода с одной страницы на другую (навигация).

Для разработки навигации воспользуемся Bootstrap, Bootstrap – CSS, HTML и JavaScript фреймворк (библиотека). Фреймворк это программный продукт упрощающий решение типовых задач и формирования архитектуры приложения. В отношении Bootstrap нет единого мнения, считать его фреймверком или библиотекой. Фреймверк, как правило диктует архитектуру приложения, в то время как Bootstrap это просто упрощения решения типовых задач. В нашем случае Bootstrap будет применен для решения типовой задачи – реализации навигации приложения.

5.1. Подключение Bootstrap

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch  
  main  
  ...  
* tc_4_4  
% git checkout -b tc_5_1
```

Switched to a new branch 'tc_5_1'

% git branch

 main

 ...

* tc_5_1

Источников информации по Bootstrap много,
например <https://getbootstrap.com/>
или например https://www.w3schools.com/bootstrap/bootstrap_ver.asp.
Или найти любой другой источник.

Возьмем самое простое, инструкцию из первой ссылки (CDN links).

The screenshot shows a browser window with the URL getbootstrap.com/docs/5.3/getting-started/introduction/. The page has a purple header with navigation links for Docs, Examples, Icons, Themes, and Blog. A search bar and a user icon are also present. The main content area is titled "2. Include Bootstrap's CSS and JS". It contains instructions: "Place the `<link>` tag in the `<head>` for our CSS, and the `<script>` tag for our JavaScript bundle (including Popper for positioning dropdowns, poppers, and tooltips) before the closing `</body>`". Below the text is a code block:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

To the right of the content area, there is a sidebar titled "On this page" with links to other sections like Quick start, CDN links, and Next steps.

В файле page.js:

The screenshot shows a code editor interface with a sidebar on the left containing project files like index.js, app.js, cat.ejs, error.ejs, index.ejs, and page.ejs. The main area displays the content of page.ejs, which includes Bootstrap imports and a script tag pointing to a CDN URL.

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hw+ALEwIH" crossorigin="anonymous">
  </head>
  </head>
  <body>
    <% body %>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLEsaAA55NDz0xhy9GkcIdsLK1eN7N6jIeHz" crossorigin="anonymous"></script>
  </body>
  </body>
  </html>
</body>
```

Bootstrap подключен и готов к использованию.

Проверим, что Bootstrap подключен и готов к использованию. На любой странице отобразим любой компонент из длинного списка возможностей Bootstrap.

Например:

The screenshot shows the Bootstrap v5.3 documentation for the Buttons component. It displays a variety of button variants (Primary, Secondary, Success, Danger, Warning, Info, Light, Dark, Link) and their corresponding HTML code. A sidebar on the left lists other components like Accordion, Alerts, Badge, Breadcrumb, Buttons, etc.

Buttons

Primary Secondary Success Danger Warning Info Light

Dark Link

HTML

```
<button type="button" class="btn btn-primary">Primary</button>
<button type="button" class="btn btn-secondary">Secondary</button>
<button type="button" class="btn btn-success">Success</button>
<button type="button" class="btn btn-danger">Danger</button>
<button type="button" class="btn btn-warning">Warning</button>
<button type="button" class="btn btn-info">Info</button>
<button type="button" class="btn btn-light">Light</button>
<button type="button" class="btn btn-dark">Dark</button>
<button type="button" class="btn btn-link">Link</button>
```

Добавляем например на главную страницу.

```
<% layout('./layout/page.ejs') %>

<h1><%= title %></h1>
<p>Welcome to <%= title %></p>
<button type="button" class="btn btn-danger">Danger</button>
```

Результат:

Express

Welcome to Express

Danger

И так, Bootstrap подключен и готов к использованию. Удаляем проверку.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_5_1
% git add .
% git commit -m'tc_5_1'
% git branch
  main
  ...
* tc_5_1
% git push origin tc_5_1
```

5.2. НАВИГАЦИОННОЕ МЕНЮ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_5_1
% git checkout -b tc_5_2
Switched to a new branch 'tc_5_2'
% git branch
  main
  ...
* tc_5_2
```

Выберем навигационное меню:

The screenshot shows the Bootstrap documentation website with a purple header. The header includes a logo, navigation links for 'Docs', 'Examples', 'Icons', 'Themes', 'Blog', a search bar, and social media links for GitHub, LinkedIn, and Twitter. Below the header, the page title is 'Navbar'. On the left, there's a sidebar with a 'Modal' heading and a 'Navbar' section selected. Other items in the sidebar include 'Navs & tabs', 'Offcanvas', 'Pagination', 'Placeholders', 'Popovers', 'Progress', 'Scrollspy', 'Spinners', 'Toasts', and 'Toolips'. The main content area contains a paragraph about the Navbar and an advertisement for Namecheap.

Modal

Navbar

Docs Examples Icons Themes Blog

Search

View on GitHub

Navbar

Documentation and examples for Bootstrap's powerful, responsive navigation header, the navbar. Includes support for branding, navigation, and more, including support for our collapse plugin.

Build your website for just \$3.88/mth. More value and performance with Namecheap.

ads via Carbon

Оставим ссылку на главный экран (Три кота) и ссылки на героев.

```
<nav class="navbar navbar-expand-lg bg-body-tertiary">
  <div class="container-fluid">
    <a class="navbar-brand" href="/">Три кота</a>
    <button class="navbar-toggler" type="button"
      data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent">
```

```

aria-controls="navbarSupportedContent"           aria-expanded="false"
aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
</button>

        <div    class="collapse    navbar-collapse"
id="navbarSupportedContent">
    <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
            <a class="nav-link" href="/korzhik">Коржик</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/karamelka">Карамелька</a>
        </li>
        <li class="nav-item">
            <a class="nav-link" href="/kompot">Компот</a>
        </li>
    </ul>
</div>
</div>
</nav>

```

Навигация добавляется в page.js. Листинг файла:

```

<!DOCTYPE html>
<html>
    <head>
        <title><%= title %></title>
        <link rel='stylesheet' href='/stylesheets/style.css' />
        <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
            rel="stylesheet"
            integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
            crossorigin="anonymous">
    </head>
    </head>
    <body>
        <nav class="navbar navbar-expand-lg bg-body-tertiary">

```

```

<div class="container-fluid">
    <a class="navbar-brand" href="/">Три кота</a>
    <button class="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0">
            <li class="nav-item">
                <a class="nav-link" href="/korzhik">Коржик</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/karamelka">Карамелька</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="/kompot">Компот</a>
            </li>
        </ul>
    </div>
</div>
</nav>
<%- body %>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz" crossorigin="anonymous"></script>
</body>
</body>
</html>
<body>

```

Проверяем:

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_5_2
% git add .
% git commit -m'tc_5_2'
% git branch
  main
  ...
* tc_5_2
% git push origin tc_5_2
```

6. Модуль MONGODB, подключение базы данных

6.1. Подключение базы данных

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_5_2
% git checkout -b tc_6_1
Switched to a new branch 'tc_6_1'
% git branch
  main
  ...
* tc_6_1
```

Откроем консоль VSCode и установим модуль mongodb.

```
npm install mongodb --save
```

```

{
  "dependencies": {
    "cookie-parser": "^1.4.4",
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "ejs-locals": "^1.0.2",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "mongodb": "^6.9.0",
    "morgan": "~1.9.1"
  }
}

```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

- yulia@MacBook-Pro-Ulia tc2024 % npm install mongodb --save

added 12 packages, and audited 69 packages in 4s

10 vulnerabilities (2 **low**, 2 **moderate**, 4 **high**, 2 **critical**)

To address issues that do not require attention, run:

 npm audit fix

To address all issues possible, run:

 npm audit fix --force

Some issues need review. and may require choosing

Переходим в IDE и в корне проекта создаем файл createDB.js.
Ссылка на синтаксис версии 6.9.0

<https://www.npmjs.com/package/mongodb/v/6.9.0-dev.20241015.sha.7fde8ddc>

Листинг файла createDB.js (для версии "mongodb": "6.9.0")

```

const { MongoClient } = require('mongodb');
// or as an es module:
// import { MongoClient } from 'mongodb'

// Connection URL
const url = 'mongodb://localhost:27017';
const client = new MongoClient(url);

```

```

// Database Name
const dbName = 'test2024';

async function main() {
  // Use connect method to connect to the server
  await client.connect();
  console.log('Connected successfully to server');
  const db = client.db(dbName);
  const collection = db.collection('documents');

  // the following code examples can be pasted here...
  const insertResult = await collection.insertMany([
    { a: 1 },
    { a: 2 },
    { a: 3 }
  ]);

  console.log('Inserted documents =>', insertResult);

  return 'done.';
}

main()
  .then(console.log)
  .catch(console.error)
  .finally(() => client.close());

```

Стартуем mongoDB

% brew services start mongodb-community

==> Successfully started `mongodb-community` (label: homebrew.mxcl.mongodb-commu ...

Стартуем mongo shell

% mongosh

Вернемся в консоль VSCode и выполним файл createDB.js:

```
$ node createDB.js
```

Проверяем базу данных:

команды для проверки	
show dbs	проверяем, что база данных test2024 создана
use test2024	переходим в базу данных test2024
db.getName()	проверяем, что находимся в базе данных test2024
db.getCollectionNames()	убеждаемся, что список коллекций состоит из одной коллекции documents
db.documents.find()	проверяем документы коллекции documents
db.dropDatabase()	удаляем базу данных test2024

Результат:

```
test> use test2024
switched to db test2024
test2024> db.getName()
test2024
test2024> db.getCollectionNames()
[ 'documents' ]
test2024> db.documents.find()
[
  { _id: ObjectId('670ecdc9ca5d927d420f5a9c'), a: 1 },
  { _id: ObjectId('670ecdc9ca5d927d420f5a9d'), a: 2 },
  { _id: ObjectId('670ecdc9ca5d927d420f5a9e'), a: 3 }
]
test2024> db.dropDatabase()
{ ok: 1, dropped: 'test2024' }
test2024>
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
...
* tc_6_1
```

```
% git add .
% git commit -m'tc_6_1'
% git branch
  main
...
* tc_6_1
% git push origin tc_6_1
```

6.2. Создание модуля данных

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_6_1
% git checkout -b tc_6_2
Switched to a new branch 'tc_6_2'
% git branch
  main
...
* tc_6_2
```

В корне создаем еще один файл, с данными для базы данных. Создадим в корне файл data.js:

```
var data = [
  {
    title: 'Карамелька',
    nick: 'karamelka',
    avatar: '/images/karamelka.jpeg',
    desc: 'Самый младший член семьи. Коронная фраза – «Я знаю, что надо делать!». Носит красный бант и красное платье.'
  },
  {
    title: 'Компот',
    nick: 'kompot',
    avatar: '/images/kompont.png',
```

```

    desc: 'Старший котёнок в семье. Ходит в школу. Очень любит грибы, читает про них энциклопедии. Любит игры на логику. Когда очень сильно огорчается, восклицает «Ну вот!» и начинает плакать. Обожает вкусно поесть. Носит зелёный костюм и шапку. ' ,
},
{
    title: 'Коржик',
    nick: 'korzhik',
    avatar: '/images/korzhik.png',
    desc: 'Средний котёнок в семье. Очень любит футбол. Любит бегать, прыгать и веселиться. Иногда делает такие вещи, что из них приходится выбираться всей семьёй. Одет в форму моряка. '
}
];

module.exports.data = data;

```

Созданный в рамках проекта js модуль подключается похожим с модулями Node.js способом. Подключим модуль data.js в файле createDB.js.

```

const { MongoClient } = require('mongodb');
// or as an es module:
// import { MongoClient } from 'mongodb'
var data = require("./data.js").data;

```

Используя вывод в консоле, убедитесь, что подготовленные данные доступны в файле createDB.js.

```

// const { MongoClient } = require('mongodb');
// // or as an es module:
// // import { MongoClient } from 'mongodb'
var data = require("./data.js").data;
console.log(data)

```

```
// // Connection URL
// const url = 'mongodb://localhost:27017';
// const client = new MongoClient(url);

// // Database Name
// const dbName = 'test2024';

// async function main() {
//   // Use connect method to connect to the server
//   await client.connect();
//   console.log('Connected successfully to server');
//   const db = client.db(dbName);
//   const collection = db.collection('documents');

//   // the following code examples can be pasted here...
//   const insertResult = await collection.insertMany([
//     { a: 1 },
//     { a: 2 },
//     { a: 3 }
//   ]);
//   console.log('Inserted documents =>', insertResult);

//   return 'done.';
// }

// main()
//   .then(console.log)
//   .catch(console.error)
//   .finally(() => client.close());
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a project structure for "TC2024" containing files like .ejs, error.ejs, index.ejs, page.ejs, package.json, createDB.js (selected), data.js, app.js, .gitignore, and several package-lock.json and package.json files.
- CODE**: The main editor area displays the content of "createDB.js". The code uses ES6 imports and MongoClient from "mongodb". It connects to a local MongoDB instance at port 27017 and logs the data object to the console. The code is as follows:

```
// const { MongoClient } = require('mongodb');
// or as an es module:
// import { MongoClient } from 'mongodb'
var data = require("./data.js").data;
console.log(data)

// Connection URL
// const url = 'mongodb://localhost:27017';
// const client = new MongoClient(url);
```

- TERMINAL**: Shows the output of a command that prints data from a JSON object. The data describes three characters: Karamelka, Kompot, and Korthik.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
...
* tc_6_2
% git add .
% git commit -m'tc_6_2'
% git branch
  main
...
* tc_6_2
% git push origin tc_6_2
```

6.3. ПОСЕВ ДАННЫХ ПРИЛОЖЕНИЯ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_6_2
% git checkout -b tc_6_3
```

```
Switched to a new branch 'tc_6_3'  
% git branch  
 main  
 ...  
* tc_6_3
```

После подключения модуля data.js в файле createDB.js, изменим название базы данных на tc2024, название коллекции на cats и передадим data в команду insertMany:

```
const { MongoClient } = require('mongodb');  
// or as an es module:  
// import { MongoClient } from 'mongodb'  
var data = require("./data.js").data;  
  
  
// Connection URL  
const url = 'mongodb://localhost:27017';  
const client = new MongoClient(url);  
  
// Database Name  
const dbName = 'tc2024';  
  
async function main() {  
  // Use connect method to connect to the server  
  await client.connect();  
  console.log('Connected successfully to server');  
  const db = client.db(dbName);  
  const collection = db.collection('cats');  
  
  // the following code examples can be pasted here...  
  const insertResult = await collection.insertMany(data);  
  console.log('Inserted documents =>', insertResult);  
  
  return 'done.';  
}  
  
main()  
.then(console.log)
```

```
.catch(console.error)
.finally(() => client.close());
```

Вернемся в консоль и выполним команду \$ node createDB.js. В консоле mongosh проверим базу данных tc2024.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_6_3
% git add .
% git commit -m'tc_6_3'
% git branch
  main
  ...
* tc_6_3
% git push origin tc_6_3
```

7. Модуль MONGOOSE, СОЗДАНИЕ МОДЕЛЕЙ ДАННЫХ

7.1. ORM

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

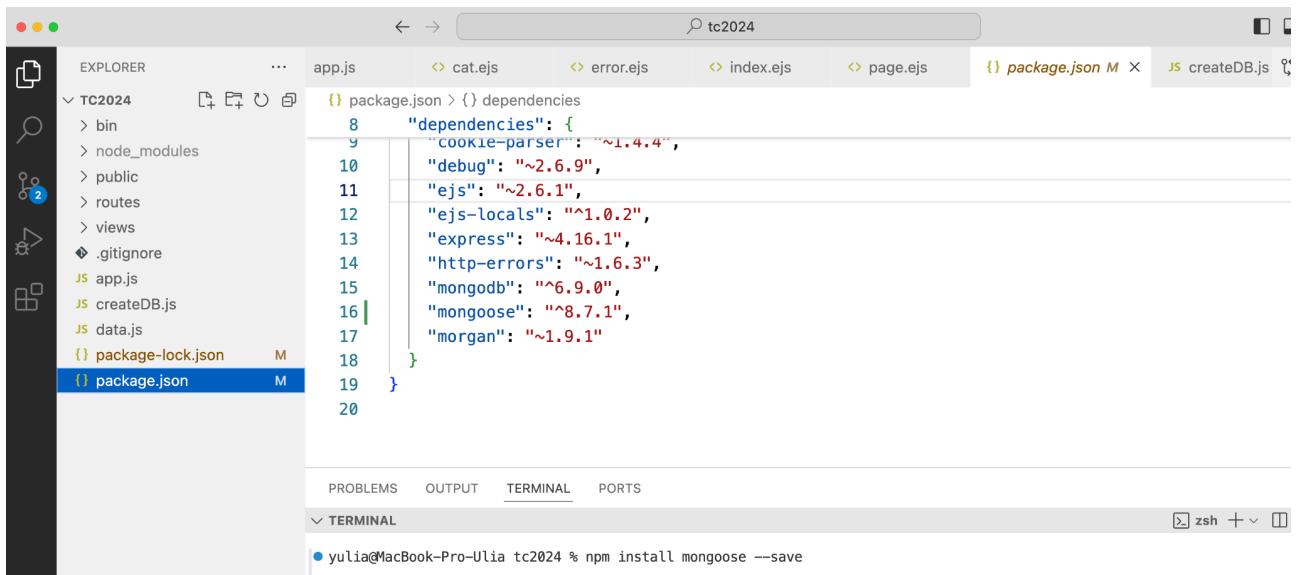
```
% git branch
  main
  ...
* tc_6_3
% git checkout -b tc_7_1
Switched to a new branch 'tc_7_1'
```

```
% git branch  
  main  
  ...  
* tc_7_1
```

Mongoose – это ORM (Object Relational Mapping) для MongoDB под Node.js. Главная задача ORM - быть связующим звеном между кодом программы и базой данных. ORM mongoose берет на себя валидацию данных, CRUD функции и бизнес-логику общения с базой данных.

В консоле установим mongoose

```
$ npm install mongoose --save
```



The screenshot shows the VS Code interface with the 'package.json' file open in the center editor. The file contains the following code:

```
8 "dependencies": {  
9   "cookie-parser": "~1.4.4",  
10  "debug": "~2.6.9",  
11  "ejs": "~2.6.1",  
12  "ejs-locals": "^1.0.2",  
13  "express": "~4.16.1",  
14  "http-errors": "~1.6.3",  
15  "mongodb": "^6.9.0",  
16  "mongoose": "^8.7.1",  
17  "morgan": "~1.9.1"  
18 }  
19 }  
20 }
```

The 'TERMINAL' tab at the bottom shows the command being run:

```
yulia@MacBook-Pro-Ulia tc2024 % npm install mongoose --save
```

Создадим новый файл testMongoose.js.

Откроем официальный сайта разработчиков mongoose

<http://mongoosejs.com/>

mongoose

elegant mongodb object modeling for node.js

[read the docs](#)

[discover plugins](#)



Star

Version 8.7.1



Let's face it, writing MongoDB validation, casting and business logic boilerplate is a drag. That's why we wrote Mongoose.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/test');

const Cat = mongoose.model('Cat', { name: String });
const kitty = new Cat({ name: 'Zildjian' });
```



Build your website for just
\$3.88/mth. More value
and performance with
Namecheap.
ads via Carbon

Запустим mongo DB:

brew services start mongodb-community

=> Successfully started `mongodb-community` (label:
homebrew.mxcl.mongodb-community)

Скопируем пример (название базы test заменим на testMongoose2024)

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/testMongoose2024');

const Cat = mongoose.model('Cat', { name: String });

const kitty = new Cat({ name: 'Пушок' });

kitty.save().then(() => console.log('Мяу'));
```

\$ node testMongoose.js

Мяу

В консоле mongo проверим базу данных test

команда	объяснение
mongosh	запуск консоли MongoDB
use testMongoose2024	переход в базу данных testMongoose2024

db.getName()	проверяем, что находимся в базе данных testMongoose2024
db.getCollectionNames()	запрос имен коллекций базы
db.cats.find()	запрос документов коллекции cats

```
|test> use testMongoose2024
switched to db testMongoose2024
|testMongoose2024> db.getName()
testMongoose2024
|testMongoose2024> db.getCollectionNames()
[ 'cats' ]
|testMongoose2024> db.cats.find()
[
  { _id: ObjectId('670f7348295ff4c71f1e19ae'), name: 'Пушок', __v: 0 }
]
|testMongoose2024> █
```

Проанализируем код.

<code>mongoose.connect('mongodb://localhost/testMongoose2024')</code>	соединение с базой данных testMongoose2024
<code>var Cat = mongoose.model('Cat', { name: String })</code>	объявление модели и полей документов этой модели
<code>var kitty = new Cat({ name: 'Пушок' })</code>	создание экземпляра модели
<code>kitty.save()</code>	сохранение данных в базе

Из имени модели mongoose формирует имя коллекции по следующему правилу – имя модели берется с маленькой буквы и преобразуется во множественное число. **Имя модели Cat – имя коллекции cats.**

Пушим ветку в удаленный репозиторий

```
% git branch
  main
```

```
...
* tc_7_1
% git add .
% git commit -m'tc_7_1'
% git branch
  main
...
* tc_7_1
% git push origin tc_7_1
```

7.2. Создание схемы

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_7_1
% git checkout -b tc_7_2
Switched to a new branch 'tc_7_2'
% git branch
  main
...
* tc_7_2
```

Реализация mongoose связи между приложением и базой данных использует два ключевых понятия – модель и схема. Вернемся в VSCode и введем в код имплементацию схемы.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://127.0.0.1:27017/testMongoose2024');

var schema = mongoose.Schema({ name: String });
const Cat = mongoose.model('Cat', schema);

const kitty = new Cat({ name: 'Пушок' });
kitty.save().then(() => console.log('Мяу'));
```

В консоле проверим результат исполнения файла и убедимся, что создание схемы просто структурировало код. Для иллюстрации возможностей схемы создадим метод схемы.

```
const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/testMongoose2024');

var schema = mongoose.Schema({ name: String })

schema.methods.meow = function() {
  console.log(this.name + " сказал мяу")
}

const Cat = mongoose.model('Cat', schema);

const kitty = new Cat({ name: 'Пушок' });
kitty.save().then(() => kitty.meow());
```

В консоле проверим код.

```
node testMongoose.js
Пушок сказал мяу
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
...
* tc_7_2
% git add .
```

```
% git commit -m'tc_7_2'  
% git branch  
  main  
  ...  
* tc_7_2  
% git push origin tc_7_2
```

7.3. Создание модели данных приложения

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch  
  main  
  ...  
* tc_7_2  
% git checkout -b tc_7_3  
Switched to a new branch 'tc_7_3'  
% git branch  
  main  
  ...  
* tc_7_3
```

Перейдем к использованию mongoose для учебного проекта. В корне проекта создадим папку models. В этой папке будем реализовывать модели приложения. Первую модель назовем Cat, поэтому в папке models создадим файл cat.js.

```
var mongoose = require('mongoose')  
var Schema = mongoose.Schema  
  
var catSchema = new Schema({  
  title: String,  
  nick: {  
    type: String,  
    unique: true,  
    required: true  
  },  
  avatar: String,  
  desc: String,  
  created:{
```

```

        type:Date,
        default:Date.now
    }
})
module.exports.Cat = mongoose.model("Cat", catSchema)

```

Проверим работу модели в файле testMongoose.js

```

const mongoose = require('mongoose');

mongoose.connect('mongodb://127.0.0.1:27017/testMongoose2024');

var Cat = require('../models/cat.js').Cat

var cat = new Cat({
    title: "Коржик"
})

cat.save();

```

Сработал валидатор:

ValidationError: Cat validation failed: nick: Path `nick` is required.

Добавим обязательное поле nick

```

var cat = new Cat({
    title: "Коржик",
    nick: "korzhik",
})

```

Проверяем

команда	объяснение
^C	остановка соединения с базой данных

```
node testMongoose.js
```

```
запуск файла testMongoose.js
```

В консоле mongosh

```
testMongoose2024> db.cats.find()
[
  { _id: ObjectId('670f7348295ff4c71f1e19ae'), name: 'Пушок', __v: 0 },
  { _id: ObjectId('670f7531869dac8fc98b973e'), name: 'Пушок', __v: 0 },
  { _id: ObjectId('670f756e394e6959bdc38c6b'), name: 'Пушок', __v: 0 },
  { _id: ObjectId('670f75eea7aef532a4b17efb'), name: 'Пушок', __v: 0 },
  { _id: ObjectId('670f77f5fa8f537dc1c22d42'), name: 'Пушок', __v: 0 },
  { _id: ObjectId('670f781bdc498d0ac037cd75'), name: 'Пушок', __v: 0 },
  {
    _id: ObjectId('670f7e6ca63464c0b7335dfb'),
    title: 'Коржик',
    nick: 'korzhik',
    created: ISODate('2024-10-16T08:50:52.179Z'),
    __v: 0
  }
]
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_7_3
% git add .
% git commit -m'tc_7_3'
% git branch
  main
  ...
* tc_7_3
% git push origin tc_7_3
```

8. ОТОБРАЖЕНИЕ ДАННЫХ В БРАУЗЕРЕ

8.1. ПОДКЛЮЧЕНИЕ БАЗЫ ДАННЫХ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_7_3
% git checkout -b tc_8_1
Switched to a new branch 'tc_8_1'
% git branch
  main
  ...
* tc_8_1
```

В шапке главного файла app.js приложения подключаем базу данных:

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var mongoose = require("mongoose")
mongoose.connect('mongodb://localhost/tc2024')
```

Проверим, что приложение работает.

команда	объяснение
npm start	запуск сервера

Проверим, что адреса /korzhik, /karamelka, /kompot отображают данные без изменения и навигация между страницами работает.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
...
```

```
* tc_8_1
% git add .
% git commit -m'tc_8_1'
% git branch
  main
...
* tc_8_1
% git push origin tc_8_1
```

8.2. ОБРАБОТКА ПАРАМЕТРА В АДРЕСЕ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_8_1
% git checkout -b tc_8_2
Switched to a new branch 'tc_8_2'
% git branch
  main
...
* tc_8_2
```

В папке routes, кроме файла index.js есть файл users.js. Это заготовка, заложенная в стартовом проекте express для обработки адресов, начинающихся со слова users. Для отображения героев создадим отдельный файл, аналогичный файлу users.js, назовем его cats.js и спроектируем роутер для адресов cats/korzhik, cats/karamelka, cats/komprot для отображения котов. Создадим в папке routes файл cats.js и скопируем в него содержимое файла users.js.

В главном файле приложения объявим новый маршрутизатор.

```
...
var routes = require('./routes/index');
var users = require('./routes/users');
var cats = require('./routes/cats');
...
```

```
app.use('/', routes);
app.use('/users', users);
app.use('/cats', cats);
```

...

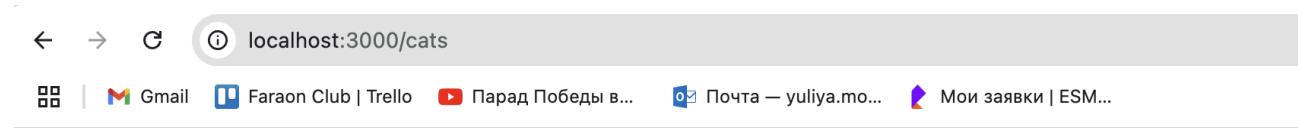
В самом файле изменим тестовый текст ответа браузеру.

```
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('Новый маршрутизатор, для маршрутов, начинающихся с cats');
});

module.exports = router;
```

Остановим и запустим сервер, проверим работу маршрута /cats.



команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Для обработки всех адресов cats/korzhik, cats/karamelka, cats/kompot одним роутером в routes/cats.js создадим роутер с параметром:

```
/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('Новый маршрутизатор, для маршрутов, начинающихся с cats');
});

/* Страница котов */
router.get("/:nick", function(req, res, next) {
  res.send(req.params.nick);
});
```

В теле маршрутизатора доступ к параметру возможен через имя параметра по следующему синтаксису: req.params.nick. Проверим что роутер видит параметр.

команда	объяснение
<code>^C</code>	остановка сервера
<code>npm start</code>	запуск сервера

localhost:3000/cats/korzhik

Gmail | Faraon Club | Trello | YouTube | Почта — yul

korzhik

localhost:3000/cats/karamelka

Нажмите, чтобы вернуться. Удерживайте, чтобы просмотреть историю.

YouTube | YouTube | Почта — yuliya.mo...

karamelka

В браузере, по адресам `cats/korzhik`, `cats/karamelka`, `cats/kompot` отображаются соответственно `korzhik`, `karamelka` и `kompot`.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_8_2
% git add .
% git commit -m'tc_8_2'
% git branch
  main
  ...
* tc_8_2
% git push origin tc_8_2
```

8.3. Извлечение данных из базы

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_8_2
% git checkout -b tc_8_3
Switched to a new branch 'tc_8_3'
% git branch
  main
  ...
* tc_8_3
```

В шапке файла роутера routes/cats.js подключим модель.

```
var express = require('express')
var router = express.Router()
var Cat = require('../models/cat').Cat;
```

Пример запроса к базе данных возьмем здесь:

[https://mongoosejs.com/docs/api/model.html#Model.find\(\)](https://mongoosejs.com/docs/api/model.html#Model.find())

The screenshot shows the Mongoose documentation page for the `Model.find()` method. The left sidebar contains links to various Mongoose features like Guides, Schemas, and Validation. The main content area has a heading "See:" followed by two bullet points: "field selection" and "query casting". Below this is a section titled "Finds documents." which explains that Mongoose casts the `filter` to match the model's schema before sending the command. It also links to a "query casting tutorial". A "Example:" section shows three lines of code demonstrating how to use `Model.find()`:

```
// find all documents
await MyModel.find({});

// find all documents named john and at least 18
await MyModel.find({ name: 'john', age: { $gte: 18 } }).exec();

// executes, name LIKE john and only selecting the "name" and "friends" fields
await MyModel.find({ name: /john/i }, 'name friends').exec();
```

В теле роутера страницы героев сделаем запрос в базе данных.

```
/* Страница котов */
```

```

router.get("/:nick", async function(req, res, next) {
  var cats = await Cat.find({nick: req.params.nick});
  console.log(cats)
  if(!cats.length) return next(new Error("Нет такого котенка в
мультфильме Три кота"))

  var cat = cats[0];
  res.render('cat', {
    title: cat.title,
    picture: cat.avatar,
    desc: cat.desc
  })
}) ;

```

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

В браузере проверим страницы: cats/korzhik, cats/karamelka, cats/kompot.

Пушим ветку в удаленный репозиторий

```

% git branch
  main

...
* tc_8_3
% git add .
% git commit -m'tc_8_3'
% git branch
  main

...
* tc_8_3
% git push origin tc_8_3

```

8.4. ПЕРЕКЛЮЧЕНИЕ АДРЕСОВ И ЧИСТКА КОДА

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_8_3
% git checkout -b tc_8_4
Switched to a new branch 'tc_8_4'
% git branch
  main
  ...
* tc_8_4
```

Страницы котов, которые открываются по адресам cats/korzhik, cats/karamelka, cats/kompot заполняются данными из базы данных. Таким образом роутеры для запросов /korzhik, /karamelka, /kompot больше не нужны. Перед тем как вычистить код от неактуальных маршрутов, в файле шаблона страницы (views/layout/page.ejs) заменим адреса страниц в навигации.

```
<li class="nav-item">
    <a class="nav-link" href="/cats/korzhik">Коржик</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="/cats/karamelka">Карамелька</a>
</li>
<li class="nav-item">
    <a class="nav-link" href="/cats/kompot">Компот</a>
</li>
```

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Используя навигацию проверим страницы: cats/korzhik, cats/karamelka, cats/komprot.

Теперь, после того как убедились, что все работает, удаляем все что было добавлено в файл routes/index.js и удалим маршрутизатор для адреса /cats.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_8_4
% git add .
% git commit -m'tc_8_4'
% git branch
  main
  ...
* tc_8_4
% git push origin tc_8_4
```

9. COOKIE и SESSION

9.1. УСТАНОВКА МОДУЛЯ EXPRESS-SESSION И НАСТРОЙКА COOKIE

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_8_4
% git checkout -b tc_9_1
Switched to a new branch 'tc_9_1'
% git branch
  main
  ...
* tc_9_1
```

Session это хранилище данных между HTTP запросами. Session создает sid — session id и помещает sid в Cookie. Cookie это текстовая информация,

которую сервер передает браузеру, браузер хранит и возвращает на сервер, для доступа к данным сессии. Самой важной частью информации cookie является sid – session id (уникальный идентификатор сессии).

Установим модуль express-session, Node.js модуль который поддерживает сохранение информации в сессии, уникальный идентификатор которой хранится в cookie.

```
$ npm install express-session --save
```

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the output of the npm install command:

```
added 7 packages, and audited 84 packages in 2s
2 packages are looking for funding
  run `npm fund` for details
10 vulnerabilities (2 low, 2 moderate, 4 high, 2 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible, run:
  npm audit fix --force
```

Настроим работу сессии. В главном файле приложения app.js подключим модуль.

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/tc2024');

var session = require("express-session")
```

Настройка работы сессии происходит через опции, которые устанавливаются в зависимости от задач, которые решает проект. В нашем учебном проекте сделаем простейшие настройки. Настройку

сессии сделаем перед объявлением роутеров, чтобы сессия существовала в момент обработки запросов.

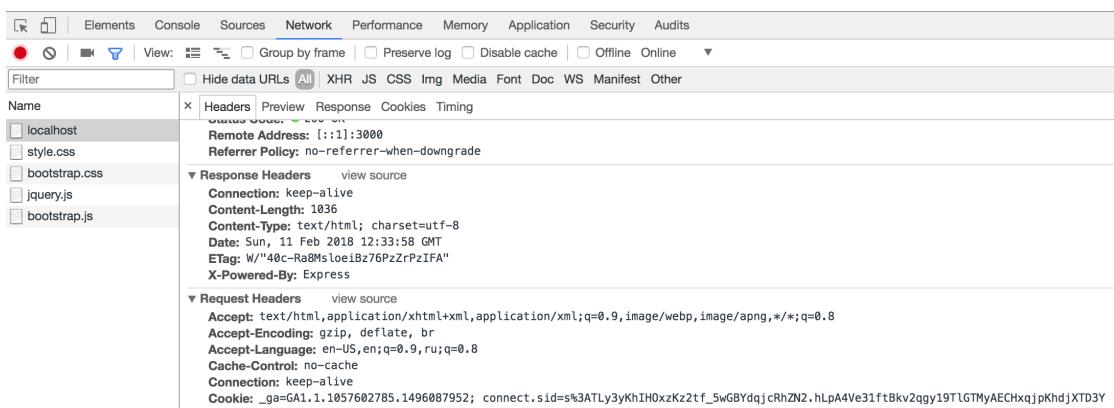
```
app.use(session({  
    secret: "ThreeCats",  
    cookie: {maxAge: 60*1000},  
    proxy: true,  
    resave: true,  
    saveUninitialized: true  
}))  
  
app.use('/', routes);  
app.use('/users', users);
```

Опция `secret` является обязательной, опция `maxAge:60*1000` определяет время жизни session в миллисекундах ($60*1000 = 1$ минута).

Запустим server.

команда	объяснение
<code>npm start</code>	запуск сервера

Перейдем в браузер `http://localhost:3000/` и найдем куки с `sid` (session id). Правой кнопкой мыши нажмем на любое место страницы в браузере и выберем инспектировать элемент.



The screenshot shows the Network tab of the Chrome DevTools. A request to 'localhost' is selected. In the 'Response' section, the 'Headers' tab is active. Under 'Headers', the 'Cookie' section displays a single cookie: `_ga=GAI.1.1057602785.1496087952; connect.sid=s%3ATLy3yKhIH0xzKz2tf_5wGBYdjqjcRhZN2.hLpA4Ve31ftBkv2qgy19TlGTMyAECHxqjpKhdjXTD3Y`.

Открываем вкладку Network => localhost => Request Headers => Cookie => `connect.sid`

Следующее место для проверки куки:

Name	Value
connect.sid	s%3ATLy3yKhlHOxzKz2tf_5wGBYdqjcRhZN2.hLpA4Ve31ftBkv2qgy19TIGTMyAECHxqjpKh...
_ga	GA1.1.1057602785.1496087952

Вкладка Application => Storage => Cookies => http://localhost:3000 => connect.sid.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_9_1
% git add .
% git commit -m'tc_9_1'
% git branch
  main
  ...
* tc_9_1
% git push origin tc_9_1
```

9.2. Команда записи в cookie

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_9_1
% git checkout -b tc_9_2
```

```
Switched to a new branch 'tc_9_2'  
% git branch  
  main  
  ...  
* tc_9_2
```

В файле routes/index.js, в обработчике запроса главной странице добавим данные в куки.

```
/* GET home page. */  
router.get('/', function(req, res, next) {  
  res.cookie('greeting', 'Hi!!!!').render('index', { title:  
    'Express' });  
});
```

Остановим сервер, запустим сервер и проверим куки в браузере.

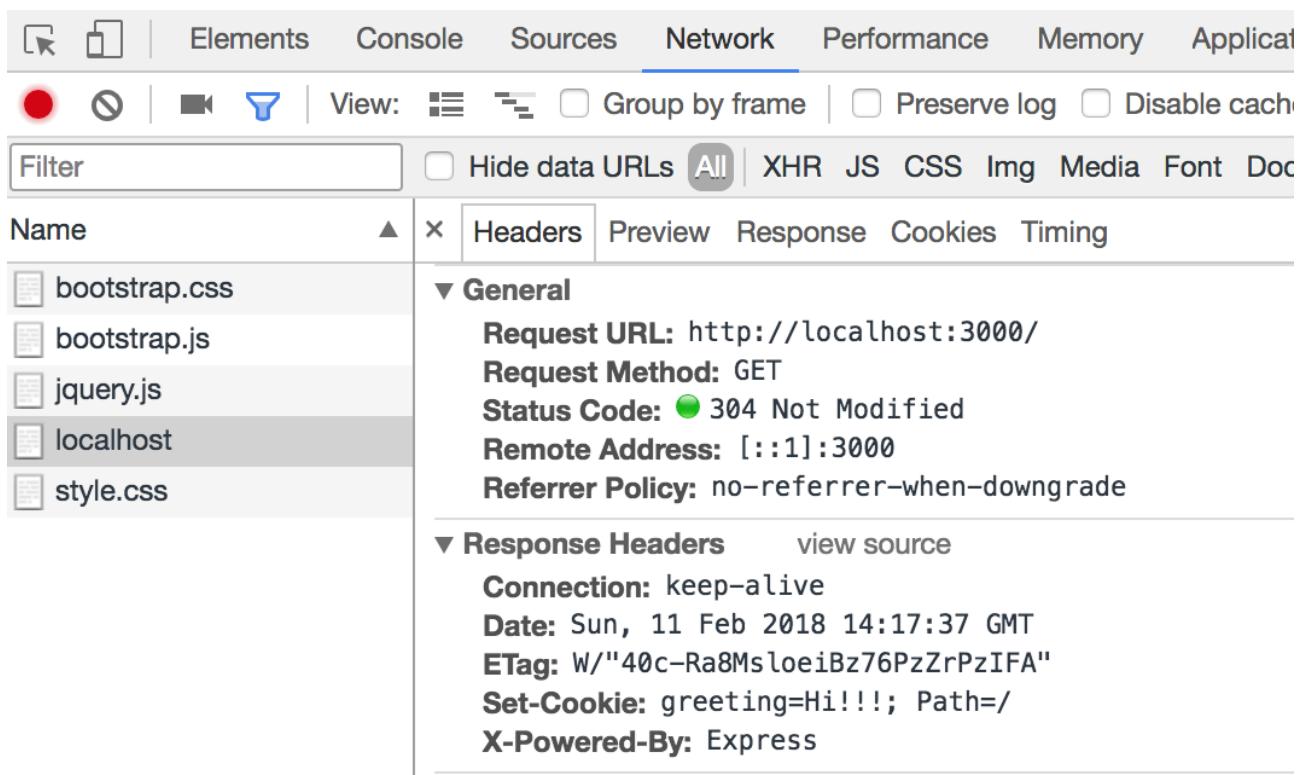
команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Во вкладке Application.

The screenshot shows the Chrome DevTools Application tab. On the left, there's a sidebar with sections for Application (Manifest, Service Workers, Clear storage), Storage (Local Storage, Session Storage, IndexedDB, Web SQL), and Cookies (under Cookies). The Cookies section shows three entries: connect.sid, greeting, and _ga. The connect.sid cookie has a long value starting with s%3AtdALPaOCdwnUP_IU1fTPZBoC7-Ui77qx.5DUDXtA6g7m162B67I%2BPlcudLBn8VOn... The greeting cookie has the value Hi!!!. The _ga cookie has the value GA1.1.1057602785.1496087952. The main pane displays a table with columns 'Name' and 'Value'.

Name	Value
connect.sid	s%3AtdALPaOCdwnUP_IU1fTPZBoC7-Ui77qx.5DUDXtA6g7m162B67I%2BPlcudLBn8VOn...
greeting	Hi!!!
_ga	GA1.1.1057602785.1496087952

Во вкладке Network.



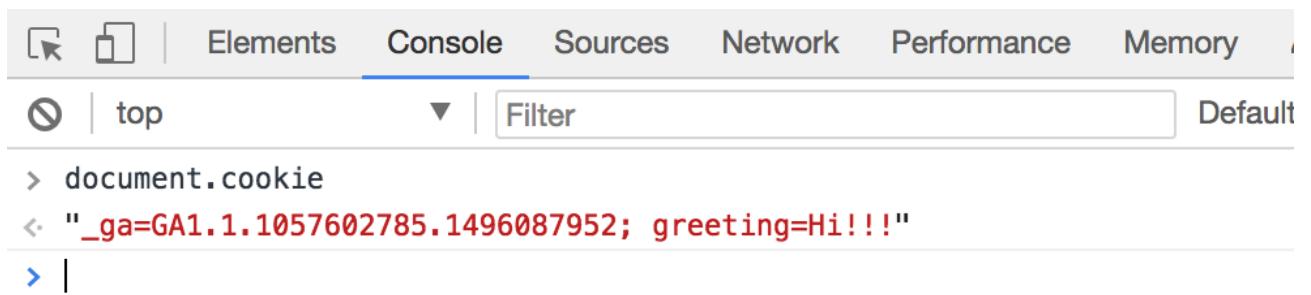
Network tab details:

- Request URL: `http://localhost:3000/`
- Request Method: GET
- Status Code: 304 Not Modified
- Remote Address: [::1]:3000
- Referrer Policy: no-referrer-when-downgrade

Response Headers:

- Connection: keep-alive
- Date: Sun, 11 Feb 2018 14:17:37 GMT
- ETag: W/"40c-Ra8MsloeiBz76PzZrPzIFA"
- Set-Cookie: greeting=Hi!!!; Path=/
- X-Powered-By: Express

В JavaScript консоле.



```
> document.cookie
< "_ga=GA1.1.1057602785.1496087952; greeting=Hi!!!"
> |
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
...
* tc_9_2
% git add .
% git commit -m'tc_9_2'
% git branch
  main
...
* tc_9_2
```

```
% git push origin tc_9_2
```

9.3. Сохранение SESSION в MongoDB

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_9_2
% git checkout -b tc_9_3
Switched to a new branch 'tc_9_3'
% git branch
  main
  ...
* tc_9_3
```

Установим модуль connect-mongo, обеспечивающий создание коллекции sessions и сохранение документов сессии.

```
$ npm install connect-mongo --save
```

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows the project structure with files like index.js, app.js, package.json, cats.js, and cat.js.
- PACKAGE.JSON**: The current file being edited, containing the following code:

```
5 "scripts": {  
6   },  
7   "dependencies": {  
8     "connect-mongo": "^5.1.0",  
9     "cookie-parser": "~1.4.4",  
10    "debug": "~2.6.9",  
11    "ejs": "~2.6.1",  
12    "ejs-locals": "^1.0.2",  
13    "express": "~4.16.1",  
14    "express-session": "^1.18.1",  
15    "http-errors": "~1.6.3",  
16    "mongodb": "^6.9.0",  
17    "mongoose": "^8.7.1",  
18    "morgan": "~1.9.1"  
19  }  
20 }
```

- TERMINAL**: Shows the command `yulia@MacBook-Pro-Ulia tc2024 % npm install connect-mongo --save` and its output: "added 7 packages, and audited 91 packages in 5s". It also indicates 2 packages are looking for funding and lists 10 vulnerabilities.

По ссылке <https://www.npmjs.com/package/connect-mongo> смотрим синтаксис подключения.

В главном файле приложения подключим модуль connect-mongo и добавим настройки сессии, для сохранения данных session в коллекции sessions.

```
var MongoStore = require('connect-mongo');  
app.use(session({  
  secret: "ThreeCats",  
  cookie: {maxAge: 60*1000},  
  proxy: true,  
  resave: true,  
  saveUninitialized: true,  
  store: MongoStore.create({mongoUrl:  
    'mongodb://localhost/tc2024'})  
}))
```

Сделаем запись в session. Для этого сделаем изменения в обработчике запроса главной страницы в файле routes/index.js.

```
/* GET home page. */  
router.get('/', function(req, res, next) {  
  req.session.greeting = "Hi!!!";
```

```
res.render('index', { title: 'Express' });
});
```

Остановим сервер, запустим сервер и проверим коллекцию sessions в Mongo Shell консоли.

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

```
|tc2024> db.getCollectionNames()
[ 'cats', 'sessions' ]
|tc2024> db.sessions.find()
[
  {
    _id: 'JLQ8W3hFaDI6KYMQ1Ix0iUVGNKs2v8us',
    expires: ISODate('2024-10-16T15:26:29.835Z'),
    session: '{"cookie":{"originalMaxAge":60000,"expires":"2024-10-16T15:26:29.835Z"
, "httpOnly":true,"path":"/"}, "greeting":"Hi!!!!"}'
  }
]
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
...
* tc_9_3
% git add .
% git commit -m'tc_9_3'
% git branch
  main
...
* tc_9_3
% git push origin tc_9_3
```

9.4. Создание счетчика посещения страниц сайта

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
```

```

main
...
* tc_9_3
% git checkout -b tc_9_4
Switched to a new branch 'tc_9_4'
% git branch
  main
...
* tc_9_4

```

В главном файле приложения app.js добавим в session переменную counter и логику счетчика посещения страниц. Код следует добавить после создания сессии, но до объявления роутеров.

```

var MongoStore = require('connect-mongo');
app.use(session({
  secret: "ThreeCats",
  cookie: {maxAge: 60*1000},
  proxy: true,
  resave: true,
  saveUninitialized: true,
  store: MongoStore.create({mongoUrl: 'mongodb://localhost/tc2024'})
}))

app.use(function(req, res, next) {
  req.session.counter = req.session.counter + 1 || 1
  next()
})

```

Остановим сервер, запустим сервер, перейдем в браузер и походим по страницам, чтобы увидеть в Mongo Shell изменение значения counter.

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

В Mongo Shell:

```
tc2024> db.sessions.find()
[  
  {  
    _id: 'wLZ3t4TX4rKVpb1bG3x3UmKwVB0mEwrf',  
    expires: ISODate('2024-10-16T15:39:21.977Z'),  
    session: '{"cookie":{"originalMaxAge":60000,"expires":"2024-10-16T15:39:21.977Z"  
,"httpOnly":true,"path":"/"}, "counter":7, "greeting":"Hi!!!"}'  
  }  
]
```

Заключительный шаг функциональности – передать counter в шаблон и отобразить counter в шаблоне. Для этого снова сделаем изменения в обработчике запроса главной страницы в файле routes/index.js.

```
/* GET home page. */  
router.get('/', function(req, res, next) {  
  res.render('index', { title: 'Express', counter:req.session.counter });  
});
```

Сделаем вывод счетчика в шаблоне главной страницы сайта. Файл views/index.ejs:

```
<% layout('./layout/page.ejs') %>  
  
<h1><%= title %></h1>  
<p>Welcome to <%= title %></p>  
<p>Счетчик: <%= counter %></p>
```

Остановим сервер, запустим сервер, перейдем в браузер, походим по страницам, чтобы увидеть на главной странице изменение значения счетчика.

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Через минуту, счетчик сбросит свое значение в 1. Это объясняется тем что значение параметра session maxAge (время жизни сессии) установлено 1 минута.

Пушим ветку в удаленный репозиторий

```
% git branch  
main  
...  
%
```

```
* tc_9_4
% git add .
% git commit -m'tc_9_4'
% git branch
  main
...
* tc_9_4
% git push origin tc_9_4
```

9.5. ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ ДЛЯ НАВИГАЦИИ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_9_4
% git checkout -b tc_9_5
Switched to a new branch 'tc_9_5'
% git branch
  main
...
* tc_9_5
```

В заключение главы рассмотрим встроенную в express возможность создания глобальных переменных для шаблонов, то есть таких переменных, которые можно использовать в любом шаблоне, не заботясь о передаче значения этой переменной в шаблон. Выбор переменных, доступных в любом шаблоне определяется логикой приложения. Глобальными стоит делать переменные, которые встречаются на каждой странице. Создадим глобальную переменную, которая будет содержать пункты навигационного меню.

В корне проекта создадим папку middlewares для посредников.
В папке middlewares создадим файл createMenu.js.

Листинг файла createMenu.js.

```
var Cat = require("../models/cat").Cat

module.exports = async function(req,res,next){
    res.locals.nav = []

    var menu = await Cat.find(null,{_id:0,title:1,nick:1});
    console.log(menu);
    if (menu.length != 0) {
        res.locals.nav = menu;
    }
    next();
}
```

nav - это глобальная переменная для массива из которого рапоним меню сайта.

Синатаксис объявления глобальной переменной шаблонов:

```
res.locals.nav
```

Посредник подготавливает массив ссылок навигационного меню используя запрос для модели Cat. Чтобы модель была доступна в файле createMenu.js, модуль модели Cat надо подключить.

Подключаем посредника в app.js.

```
app.use(function(req,res,next){
    req.session.counter = req.session.counter + 1 || 1
    next()
})

app.use(require("./middlewares/createMenu"))

app.use('/', indexRouter);
app.use('/users', usersRouter);
```

```
app.use('/cats', cats);
```

The screenshot shows the Visual Studio Code interface. The code editor displays a file named `createMenu.js` with the following content:

```
middlewares > JS createMenu.js > <unknown> > exports
1 var Cat = require("../models/cat").Cat
2
3
4 module.exports = async function(req,res,next){
5     res.locals.nav = []
6
7     var menu = await Cat.find(null,{_id:0,title:1,nick:1});
8     console.log(menu);
9     if (menu.length) {
10         res.locals.nav = menu;
11     }
12     next();
13 }
```

The terminal below shows the application starting and displaying some data:

```
Node.js v22.8.0
yulia@MacBook-Pro-Ulia tc2024 % npm start
> tc2024@0.0.0 start
> node ./bin/www

[
  { title: 'Карамелька', nick: 'karamelka' },
  { title: 'Компот', nick: 'kompot' },
  { title: 'Коржик', nick: 'korzhik' }
]
GET / 200 23.931 ms - 1694
GET /stylesheets/style.css 304 2.210 ms - -
```

Перейдем к использованию переменной `nav` в `layout` сайта: `views/layout/page.ejs`.

```
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
  <% if(typeof nav == 'object' && nav) {
    nav.forEach(function(item) {
      %>
      <li class="nav-item">
        <a class="nav-link" href="/cats/<%= item.nick %>"><%= item.title %></a>
      </li>
      <%
    }) %>
  </ul>
```

Проверим, что приложение работает с глобальной переменной nav.

команда	объяснение
^C	остановка сервера
npm start	запуск сервера

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_9_5
% git add .
% git commit -m'tc_9_5'
% git branch
  main
  ...
* tc_9_5
% git push origin tc_9_5
```

10. АУТЕНТИФИКАЦИЯ

10.1. Создание страницы регистрации

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_9_5
% git checkout -b tc_10_1
Switched to a new branch 'tc_10_1'
% git branch
  main
  ...
* tc_10_1
```

К стандартной функциональности web-проекта относится

аутентификация пользователя. Под аутентификацией понимают вход пользователя в закрытую систему. Кроме термина аутентификация существует термин авторизация, как правило под авторизацией понимают определение прав пользователя. Добавим в учебный проект функциональность аутентификации.

Добавим к проекту страницу с регистрационной формой. Для этого:

1. Добавим роутер новой страницы.
2. Создадим шаблон с формой.
3. Добавим в навигационной меню ссылку на страницу регистрации.

В файле routes/index.js добавим новый роутер.

```
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  req.session.greeting = "Hi!!!";
  res.render('index', { title: 'Express',
counter:req.session.counter });
});

/* GET login/registration page. */
router.get('/logreg', function(req, res, next) {
  res.render('logreg',{title: 'Вход'});
});

module.exports = router;
```

По адресу /logreg браузер откроет страницу шаблона logreg. Создадим шаблон logreg.ejs в папке views. В начале страницы регистрационной формы подключим шаблон страницы с навигационным меню.

```
<% layout('/layout/page.ejs') %>
```

Для верстки формы воспользуемся готовым Bootstrap решением:

```
<div class="conteiner">
  <h1>Регистрация и вход</h1>
  <p>Введите имя пользователя и пароль, если такого пользователя нет, то он будет создан.</p>
  <form class="form-horizontal">
    <div class="form-group">
      <label class="control-label col-sm-1">Имя</label>
      <div class="col-sm-5">
        <input class="form-control" type="text"
              placeholder="Введите имя"/>
      </div>
    </div>
    <div class="form-group">
      <label class="control-label col-sm-1">Пароль</label>
      <div class="col-sm-5">
        <input class="form-control" type="password"
              placeholder="Введите пароль"/>
      </div>
    </div>
    <div class="form-group">
      <div class="col-sm-offset-1 col-sm-5">
        <input class="btn btn-primary" type="submit"
              value="Войти"/>
      </div>
    </div>
  </form>
</div>
```

Отметим, что кроме классов Bootstrap отвечающих за стиль формы в верстке используются классы Bootstrap Grid. Bootstrap Grid это разбивка страницы браузера на 12 колонок. При этом Bootstrap Grid использует 4 класса xs, sm, md, lg.

класс	ширина экрана для применения стиля класса
xs	ширина экрана меньше 768px
sm	ширина экрана меньше 992px

	больше или равна 768px
md	ширина экрана меньше 1200px больше или равна 992px
lg	больше или равна 1200px

Например, класс col-sm-1 для тега <label> определяет стиль <label>, в зависимости от ширины экрана следующим образом: При ширине экрана больше или равной 768px тег <label> занимает 1 колонку. При ширине экрана меньше 768px количество колонок определяется классом xs, если класс xs для <label> не задан, то <label> занимает 12 колонок. При ширине экрана больше или равна 992px количество колонок, которые занимает <label> определяется классом md. Если класс md для <label> не задан, количество колонок остается равным настройке предыдущего размера экрана.

Для доступа к странице регистрации осталось добавить пункт навигационного меню. В шаблоне страницы views/layout/page.ejs добавим ссылку «Войти».

```
<ul class="nav navbar-nav navbar-right">
  <li class="nav-item">
    <a class="nav-link" href="/logreg">Войти</a>
  </li>
</ul>
```

Проверим, что в навигационном меню появилась новая ссылка «Войти», после нажатия на которую открывается форма с полями регистрации.

Три кота Карамелька Компот Коржик

Войти

Express

Welcome to Express

Счетчик: 3

Три кота Карамелька Компот Коржик

Войти

Регистрация и вход

Введите имя пользователя и пароль, если такого пользователя нет, то он будет создан.

Имя

name

Пароль

.....

Войти

Три кота



Регистрация и вход

Введите имя пользователя и пароль, если такого пользователя нет, то он будет создан.

Имя

name

Пароль

.....

Войти

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_1
% git add .
% git commit -m'tc_10_1'
% git branch
  main
  ...
* tc_10_1
% git push origin tc_10_1
```

10.2. Создание модели User

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_10_1
```

```
% git checkout -b tc_10_2
Switched to a new branch 'tc_10_2'
% git branch
  main
...
* tc_10_2
```

Для управления пользователем приложения создадим mongoose модель. Назовем модель User, mongoose создаст в базе данных коллекцию users. Синтаксис создания mongoose модели мы уже обсуждали при создании модели хранения данных Cat.

Для хранения моделей в проекте в Главе 8 была создана специальная папка models. Структурирование проекта является частью хорошего стиля программирования. Название папки подчеркивает назначение папки. Папка models самим названием определяет содержимое. Для создания модели пользователя добавим в папку models файл user.js.

В начале файла модели подключим модуль mongoose.

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema

var userSchema = new Schema({
  username: {
    type: String,
    unique: true,
    required: true
  },
  hashedPassword: {
    type: String,
    required: true
  },
  salt: {
    type: String,
    required: true
  },
  created: {
    type: Date,
    default: Date.now
  }
})
```

```

    }
})

module.exports.User = mongoose.model("User", userSchema)

```

Проанализируем поля схемы. Поле `username` это имя пользователя, которое будет введено пользователем в первое поле формы на странице регистрации. Второе поле регистрационной формы – пароль. Но среди полей схемы нет поля `password`. Отсутствие поля `password` в коллекции базы данных является хорошим стилем программирования. Не принято хранить в базах данных пароли пользователей, а принято хранить хеш отпечатки паролей, усиленные для безопасности случайной добавкой, которая обычно носит название соль. Поэтому в схеме заданы поля `hashedPassword` и `salt`. Поле `created` фиксирует дату создания пользователя.

Для работы с паролем добавим в схему виртуальное поле `password`. Виртуальное поле – это поле которое не сохраняется в базе данных, но используется для вычисления других полей, в нашем случае `hashedPassword` и `salt`.

```

userSchema.virtual("password").set(function(password){
  this._purePassword = password
  this.salt = Math.random() + ""
  this.hashedPassword = this.encryptPassword(password)
}).get(function(){
  return this._purePassword
})

userSchema.methods.encryptPassword = function(password){
  return crypto.createHmac('sha1', this.salt).update(password).digest('hex')
}

```

Помещаем перед строкой:

```
module.exports.User = mongoose.model("User", userSchema)
```

Создание `hashedPassword` вынесено в отдельный метод схемы. Для поддержки криптографических методов установим и подключим метод `crypto` в шапке файла `User.js`.

```

var mongoose = require("mongoose")
var crypto = require("crypto")

```

Проверим работу модели. Для проверки работы модели, в корне проекта, создадим временный файл checkUser.js. После тестирования модели файл checkUser.js можно удалить.

Листинг файла checkUser.js

```
var mongoose = require("mongoose")
mongoose.connect("mongodb://localhost/tc2024")
var User = require("./models/user.js").User

var first_user = new User({
    username: "Vasya",
    password: "qwerty"
})

first_user.save();
```

В файле подключаем mongoose модуль, подключаемся к базе данных, подключаем модель User. Создание пользователя происходит в два шага.

Шаг 1: Создание экземпляра модели со значениями имени пользователя и пароля.

Шаг 2: Сохранение пользователя.

Выполним файл checkUser.js и проверим в базе данных результат.

команда	объяснение
node checkUser.js	выполнение файла checkUser.js
mongo	запуск Mongo Shell
use all	переход в базу данных all
db.getCollectionNames()	список коллекций в базе
db.users.find()	запрос документов коллекции users

```
|tc2024> db.getCollectionNames()
[ 'cats', 'sessions', 'users' ]
|tc2024> db.users.find()
[
  {
    _id: ObjectId('670ff0a7c337092f424d8876'),
    username: 'Vasya',
    hashedPassword: 'f8e03075ddfbb358d0db392547ca97faef3a07bf',
    salt: '0.5445705749350143',
    created: ISODate('2024-10-16T16:58:15.761Z'),
    __v: 0
  }
]
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_2
% git add .
% git commit -m'tc_10_2'
% git branch
  main
  ...
* tc_10_2
% git push origin tc_10_2
```

10.3. ПОДГОТОВКА ДАННЫХ ДЛЯ ПЕРЕДАЧИ НА СЕРВЕР

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_10_2
% git checkout -b tc_10_3
Switched to a new branch 'tc_10_3'
% git branch
  main
  ...
* tc_10_3
```

Перейдем к этапу передачи данных со страницы формы в роутер. Отметим атрибуты формы, которые обеспечивают передачу данных.

```
<form class="form-horizontal" action="/logreg" method="post">
  <div class="form-group">
    <label class="control-label col-sm-1">Имя</label>
    <div class="col-sm-5">
      <input class="form-control" type="text"
            placeholder="Введите имя" name="username"/>
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-sm-1">Пароль</label>
    <div class="col-sm-5">
      <input class="form-control" type="password"
            placeholder="Введите пароль" name="password"/>
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-1 col-sm-5">
      <input class="btn btn-primary" type="submit"
            value="Войти"/>
    </div>
  </div>
</form>
```

атрибут	объяснение
action="/logreg"	адрес отправки данных
method="post"	метод http запроса
name="username"	имя параметра поля ввода имени
name="password"	имя параметра поля ввода пароля
type="submit"	тригер для отправки данных

В файле роутера routes/index.js добавим обработчик для метода post:

```
/* POST login/registration page. */
router.post('/logreg', function(req, res, next) {
  });

});
```

Для обеспечения получения данных в роутере-обработчике в Node.js предусмотрен модуль body-parser. В стартовом проекте express этот

модуль уже подключен в главном файле приложения app.js. После чего, получение значений параметров методом пост примет вид:

```
var username = req.body.username  
var password = req.body.password
```

Для проверки получения логина и пароля с формы регистрации распечатаем параметры формы.

```
/* POST login/registration page. */  
router.post('/logreg', function(req, res, next) {  
    var username = req.body.username  
    var password = req.body.password  
    console.log(username);  
    console.log(password);  
});
```

Выполнить проверку.

Пушим ветку в удаленный репозиторий

```
% git branch  
  main  
  ...  
* tc_10_3  
% git add .  
% git commit -m'tc_10_3'  
% git branch  
  main  
  ...  
* tc_10_3  
% git push origin tc_10_3
```

10.4. ЛОГИКА ПОЛЬЗОВАТЕЛЯ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_10_3
% git checkout -b tc_10_4
Switched to a new branch 'tc_10_4'
% git branch
  main
  ...
* tc_10_4
```

Логика пользователя включает прежде всего проверку существования пользователя с именем `username` в базе данных. В шапке роутера (`routes/index.js`) подключим модель `User`.

```
var express = require('express');
var router = express.Router();
var User = require('../models/user').User;
```

В обработчике метода `post` подготовим логическую схему первого шага анализа `username` и `password`.

```
/* POST login/registration page. */
router.post('/logreg', async function(req, res, next) {
  var username = req.body.username
  var password = req.body.password
  console.log(username);
  console.log(password);
  var users = await User.find({username: username});
  console.log(users);
  if (!users.length) {
    res.send("<h1>Пользователь НЕ найден</h1>");
  }
});
```

```

    } else {
      res.send("<h1>Пользователь найден</h1>");
    }

  });

```

Для случая «Пользователь НЕ найден» реализуем создание нового пользователя.

```

//res.send("<h1>Пользователь НЕ найден</h1>");

var user = new User({username:username,password:password})
await user.save();
req.session.user_id = user._id;
res.redirect('/');

```

В случае успешного создания нового пользователя в переменную сессии user_id сохраним _id нового пользователя. Заключительная строчка логики – res.redirect('/') – это переход на главную страницу после успешного создания нового пользователя.

В случае существования пользователя с именем username подготавим логическую схему второго шага:

Функцию checkPassword создадим как метод схемы модели User. Для этого в файл модели models/user.js добавим функцию checkPassword.

```

userSchema.methods.checkPassword = function(password) {
  return this.encryptPassword(password) === this.hashedPassword
}

```

В результате логика пользователя примет вид:

```

/* POST login/registration page. */
router.post('/logreg', async function(req, res, next) {
  var username = req.body.username

```

```

var password = req.body.password
console.log(username);
console.log(password);
var users = await User.find({username: username});
console.log(users);
if (!users.length) {
    //res.send("<h1>Пользователь НЕ найден</h1>");
    var user = new
User({username:username,password:password})
        await user.save();
req.session.user_id = user._id;
res.redirect('/');
} else {
    //res.send("<h1>Пользователь найден</h1>");
    var foundUser = users[0];
    if(foundUser.checkPassword(password)){
        req.session.user_id = foundUser._id
        res.redirect('/')
    } else {
        res.render('logreg',{title: 'Вход'});
    }
}
});
```

В случае правильного пароля в сессию сохраняем _id пользователя и переходим на главную страницу. В случае неправильного пароля возвращаем пользователя на страницу регистрации. Если пароль не верный, то пользователь не просто должен быть возвращен на страницу регистрации, а получить сообщение о неправильном пароле, то есть получить сообщение валидации. Логика сообщения о неправильном пароле обязательно будет добавлена позже.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_4
% git add .
% git commit -m'tc_10_4'
% git branch
  main
  ...
* tc_10_4
% git push origin tc_10_4
```

10.5. ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ USER

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_10_4
% git checkout -b tc_10_5
Switched to a new branch 'tc_10_5'
% git branch
  main
  ...
* tc_10_5
```

Создадим глобальную переменную user, которая будет доступна во всех шаблонах, по аналогии с созданием глобальной переменной nav для навигационного меню.

В папке middleware создадим файл createUser.js.

Листинг файла createUser.js

```

var User = require("../models/user").User;

module.exports = async function(req, res, next) {
  res.locals.user = null

  var users = await User.findById(req.session.user_id);
  if (users.length != 0) {
    res.locals.user = users[0];
  }
  next();
}

```

Подключение модели, поиск пользователя по `_id` пользователя, сохраненного в сессии, создание глобальной переменной `user` и `exports` middleware функции.

Подключим созданный middleware в главном файле приложения `app.js`.

```

app.use(require("./middlewares/createMenu.js"))
app.use(require("./middlewares/createUser.js"))

app.use('/', routes);
app.use('/users', users);

```

Переменную `user` можно использовать в любом шаблоне.

Воспользуемся переменной `user` на главной странице сайта. Откроем файл `views/index.ejs` и добавим JavaScript код.

```

<% if(user) {%
  // Случай залогиненного пользователя
<% }else{ %
  // Случай не залогиненного пользователя
<% } %>

```

Листинг файла `views/index.ejs`

```

<% layout('./layout/page.ejs') %>

<% if(user) {%
  <h1>Привет <%= user.username %></h1>
<% }else{ %

```

```
<h1>Привет незалогиненный пользователь</h1>
<% } %>
```

```
<h1><%= title %></h1>
<p>Welcome to <%= title %></p>
<p>Счетчик: <%= counter %></p>
```

Для случая залогиненного пользователя добавим приветствие пользователя по имени. Для случая незалогиненного пользователя добавим рекомендацию залогиниться для просмотра страниц сайта.

Три кота



Привет незалогиненный пользователь Express

Welcome to Express

Счетчик: 1

localhost:3000

Gmail Faraon Club | Trello Парад Победы в... Почта — yuliya.mo... Мои заявки | ESM...

Все закладки

Три кота

Карамелька

Компот

Коржик

Войти

Привет незалогиненный пользователь Express

Welcome to Express

Три кота

Регистрация и вход

Введите имя пользователя и пароль, если такого пользователя нет, то он будет создан.

Имя

Пароль

Войти

Три кота



Привет name Express

Welcome to Express

Счетчик: 4

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_5
% git add .
% git commit -m'tc_10_5'
% git branch
  main
  ...
* tc_10_5
% git push origin tc_10_5
```

10.6. ОБРАБОТКА ОШИБКИ АУТЕНТИФИКАЦИИ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
```

```
main
...
* tc_10_5
% git checkout -b tc_10_6
Switched to a new branch 'tc_10_6'
% git branch
  main
...
* tc_10_6
```

На странице регистрационной формы добавим <div> для отображения ошибки аутентификации:

```
...
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-8">
    <input class="btn btn-primary" type="submit"
           placeholder="" />
  </div>
</div>
<div class="alert alert-danger">Пароль не верный</div>
</form>
```

Bootstrap классы alert и alert-danger обеспечивают стиль сообщения об ошибке.

The screenshot shows a registration/login form with the following elements:

- Header navigation: Три кота, Карамелька, Компот, Коржик, Войти
- Section title: Регистрация и вход
- Text instructions: Введите имя пользователя и пароль, если такого пользователя нет, то он будет создан.
- Input field: Имя (placeholder: name)
- Input field: Пароль (placeholder:)
- Submit button: Войти
- Error message: Пароль не верный (displayed in a red box at the bottom)

Шаблон регистрационной формы вызывается в приложении дважды. Первый раз, когда открываем страницу регистрационной формы при нажатии на ссылку «Войти» и второй раз, когда возвращаем

пользователя на страницу регистрационной формы после ошибки аутентификации. Передадим в шаблон переменную error. Для первого вызова шаблона

```
/* GET login/registration page. */
router.get('/logreg', function(req, res, next) {
  res.render('logreg',{title: 'Вход', error: null});
});
```

переменной error присваиваем значение null.

Для второго вызова шаблона

```
//res.send("<h1>Пользователь найден</h1>");

var foundUser = users[0];
if(foundUser.checkPassword(password)) {
  req.session.user_id = foundUser._id
  res.redirect('/')
} else {
  res.render('logreg',{title: 'Вход', error: 'Пароль не
верный'});
}
```

переменной error присваивается сообщение об ошибке.

Добавим обработку переменной error в шаблоне.

```
...
<div class="form-group">
  <div class="col-sm-offset-2 col-sm-8">
    <input class="btn btn-primary" type="submit"
placeholder="">
  </div>
</div>
<% if(error) { %>
  <div class="alert alert-danger"><%= error %></div>
<% } %>
</form>
```

Пушим ветку в удаленный репозиторий

```
% git branch
```

```
main
...
* tc_10_6
% git add .
% git commit -m'tc_10_6'
% git branch
  main
...
* tc_10_6
% git push origin tc_10_6
```

10.7. ФУНКЦИОНАЛЬНОСТЬ LOGOUT

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
...
* tc_10_6
% git checkout -b tc_10_7
Switched to a new branch 'tc_10_7'
% git branch
  main
...
* tc_10_7
```

Добавим в приложение функциональность logout. Для этого в шаблоне навигационного меню добавим отображение ссылки «Выйти» для залогиненного пользователя. Перейдем в файл views/logout/page.js.

```
<ul class="nav navbar-nav navbar-right">
<%if(user){%>
  <form action="/logout" method="post">
    <li class="nav-item">
      <button type="submit" class="nav-link">Выйти</button>
    </li>
```

```
</form>

<% } else { %>
    <li class="nav-item">
        <a class="nav-link" href="/logreg">Войти</a>
    </li>
<% } %>
</ul>
```

В файле routes/index.js добавим post роутер для адреса /logout.

```
/* POST logout. */
router.post('/logout', function(req, res, next) {
    req.session.destroy();
    res.locals.user = null;
    res.redirect('/');
});
```

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_7
% git add .
% git commit -m'tc_10_7'
% git branch
  main
  ...
* tc_10_7
% git push origin tc_10_7
```

10.8. ЗАКРЫТИЕ СТРАНИЦ САЙТА ДЛЯ НЕЗАЛОГИНЕННОГО ПОЛЬЗОВАТЕЛЯ

Перед началом работы с пунктом создаем новую ветку и переходим работать на новую ветку.

```
% git branch
  main
  ...
* tc_10_7
% git checkout -b tc_10_8
Switched to a new branch 'tc_10_8'
% git branch
  main
  ...
* tc_10_8
```

Последним штрихом завершения учебного проекта реализуем функциональность закрытия страниц сайта для незалогиненного пользователя. Для ограничения доступа к определенным страницам сайта воспользуемся технологией middleware. В пособии к возможностям посредника мы обращались дважды. Первый раз для создания глобальной переменной nav и второй раз для создания глобальной переменной user. Третий раз повторим создание посредника для проверки авторизованного пользователя.

В папку `middleware` добавим новый файл `checkAuth.js`.

Листинг файла checkAuth.js

```
module.exports = function(req, res, next) {
  if(!req.session.user_id) {
    res.redirect("/")
  } else {
    next()
  }
}
```

Логика реализованная в посреднике очень простая: если есть `_id` (если пользователь залогиненный), то посредник запускает функцию `next()` которая позволяет следовать логике приложения дальше, если `_id` в сессии не существует, то происходит редирект на главную страницу сайта, где для незалогиненного пользователя отображается рекомендация залогиниться, чтобы увидеть все страницы сайта.

Подключим посредник checkAuth.js в шапке файла routes/cats.js.

```
var express = require('express');
var router = express.Router();
var Cat = require('../models/cat').Cat;
var checkAuth = require("../middlewares/checkAuth.js");
```

Для ограничения доступа к страницам героев добавим посредник вторым аргументом в роутер.

```
/* Страница котов */
router.get("/:nick", checkAuth, async function(req, res, next) {
...
})
```

Проверим, что аутентификация и авторизация пользователя работает как было задумано в учебном проекте.

Пушим ветку в удаленный репозиторий

```
% git branch
  main
  ...
* tc_10_8
% git add .
% git commit -m'tc_10_8'
% git branch
  main
  ...
* tc_10_8
% git push origin tc_10_8
```