

ОСНОВНАЯ ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА ПОДГОТОВКИ БАКАЛАВРА

по направлению

09.03.01 Информатика и вычислительная техника профиль
"Технологии разработки программного обеспечения"

Б. 1.10.2 Модуль "Проектирование и разработка веб-решений".
Компьютерный практикум

Приложение 1

Типовые задания для проведения процедур оценивания результатов освоения дисциплины в ходе текущего контроля, шкалы и критерии оценивания

Содержание

1. [Типовые задания лабораторных работ по темам](#)
2. [Типовые задания для инвариантной самостоятельной работы по темам](#)
3. [Типовые задания для вариативной самостоятельной работы по темам](#)

1. Типовые задания лабораторных работ по темам

Задания лабораторных работ направлены на отработку действий, способов, методов решения задач в области веб-проектирования и веб-дизайна, которыми обучающиеся должны владеть.

Задания для лабораторных работ предполагают преимущественно репродуктивный характер действий обучающихся и рассчитаны на выполнение в рамках определенного количества часов лабораторных занятий (Таблица 3). Задания представлены в СДО Moodle в электронном учебном курсе по дисциплине «Компьютерный практикум» и могут иметь одну или несколько из следующих форм:

- краткая формулировка постановки задачи с пояснениями;
- пошаговая инструкция;
- скринкаст.

Далее приводятся ссылки на веб-страницы на сервере Кодактор.ру (например, https://kodaktor.ru/g/xml_intro), содержащие более детальные инструкции к заданиям и поясняющие скринкасты.

Выполнение задания предполагает следующие виды деятельности:

- разработку сценария или приложения на языке разметки XML или веб-языке (языке программирования высокого уровня в области веб-ресурсов):
 - JavaScript;
 - и, возможно, другом языке по выбору обучающихся;
- выявление и исправление синтаксических ошибок, выполнение транспиляции, тестирования работы сценария.
- составление отчета о выполненном задании в виде слайдов и/или онлайн-документации, размещаемой в Git-репозитории как части веб-портфолио обучающегося (https://github.com/GossJS/report_template и https://kodaktor.ru/g/report_template).

Критерий оценивания. Лабораторная работа считается выполненной, если веб-сценарий или веб-приложение разработаны и соответствуют заданию, проходят процедуры тестирования, а также сопровождается репозиторием (в том числе, возможно, отчетом в

форме слайдов). Часть заданий проверяется (или дополнительно проверяется) с помощью средств автоматизированной проверки, отсылающей запросы к веб-сценариям. Веб-сценарии должны быть размещены на ресурсе, допускающем обращение через Интернет (онлайн-редактор типа Кодактор.ру или coderepen.io):

<https://kodaktor.ru/g/z7a>

Автопроверка требования z7a

<h4 id="author" title="GossJS">Иван Иванов</h4> должны помещаться имя и фамилия исполнителя

В случае серверных приложений имя и фамилия исполнителя должны выдаваться ещё и по маршруту GET /author
Выдача по маршруту /author должна быть с CORS

URL

GossJS? Author

/author

Выполнение заданий самого простого вида в Кодактор.ру: <https://kodaktor.ru/dzjs>

Тема 1. Структура экосистемы веб-языков и технологий.

I. Развёртывание проекта на JavaScript, включающего модули

1. Убедитесь что установлены node, npm и yarn (https://kodaktor.ru/js01_intro_lr.pdf)
2. Создайте новый проект: `mkdir $(date +%Y%m%d_%H%M%S) && cd $_ && yarn init -y` или `mkdir $(date +%Y%m%d_%H%M%S) && cd $_ && npm init -y` (<https://kodaktor.ru/g/init>)
3. Добавьте зависимости babel-cli и babel-preset-env в раздел девелоперских зависимостей `yarn add --dev babel-cli babel-preset-env`
4. Создайте простейшую настройку babel в файле .babelrc `echo '{"presets":["env"]}' > .babelrc`
5. Добавьте к проекту библиотеку moment для работы с датами/временем. `yarn add moment`
6. Разместите файлы, находящиеся в отношении нативной модульности, в папке ./src

файл	содержимое
main.js	<pre>import moment from 'moment'; import name from './name'; console.log(name); console.log(moment.unix('1500514362').format('DD.MM.YYYY HH:mm:ss'));</pre>
name.js	<pre>export default 'E. B. Goss';</pre>

7. Выполните команду транспиляции:

`npx babel ./src -d ./lib`

Опционально можно создать соответствующий сценарий в package.json (без npx).

```
"scripts": {
  "trans": "babel-node ./src/index.js",
  "build": "babel ./src -d ./lib"
},
```

и выполнить `yarn run build`

8. Убедитесь, что получена папка lib с транспирированными файлами, в которых вместо import и export будут require и module.exports и что вызов файла main.js с помощью node не вызывает ошибок:

```
$ npx babel ./src -d ./lib
src/main.js -> lib/main.js
src/name.js -> lib/name.js
-> eliasgoss@ ~/20180212_001921
$ ls
index.js  lib  node_modules  package.json  result.js  src
-> eliasgoss@ ~/20180212_001921
$ ls ./lib
main.js name.js
-> eliasgoss@ ~/20180212_001921
$ node ./lib/main
E. B. Goss
20.07.2017 04:32:42
```

пофайловое преобразования

успешное исполнение транспирированного кода

9. Сверьте содержимое полученных файлов с таблицей:

файл	содержимое
main.js	<pre>'use strict'; var _moment = require('moment'); var _moment2 = _interopRequireDefault(_moment); var _name = require('./name'); var _name2 = _interopRequireDefault(_name); function _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { default: obj }; } console.log(_name2.default); console.log(_moment2.default.unix('1500514362').format('DD.MM.YYYY HH:mm:ss'));</pre>
name.js	<pre>'use strict'; Object.defineProperty(exports, "__esModule", { value: true }); exports.default = 'E. B. Goss';</pre>

10. Разместите полученный отчёт (лог действий) в репозитории (веб-портфолио).

II. Преобразование кода на JavaScript, содержащего конструкции из следующих версий стандарта ECMAScript

Осуществите преобразование кода, содержащего поля класса. Для этого:

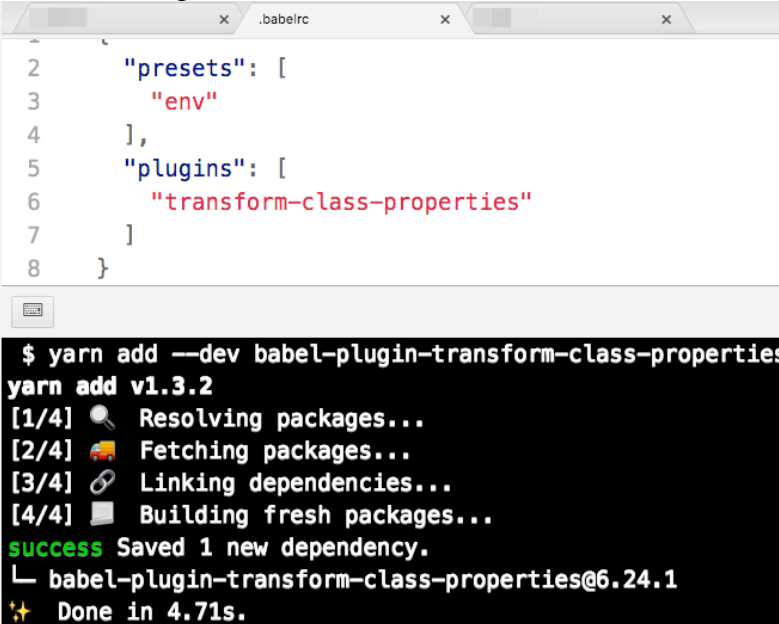
1. Добавьте поддержку class properties

```
yarn add --dev babel-plugin-transform-class-properties
```

или

```
npm i -D babel-plugin-transform-class-properties
```

2. Измените файл .babelrc



The screenshot shows a code editor with a file named .babelrc. The file content is as follows:

```
2   "presets": [  
3     "env"  
4   ],  
5   "plugins": [  
6     "transform-class-properties"  
7   ]  
8 }
```

Below the editor is a terminal window showing the command to install the plugin and its output:

```
$ yarn add --dev babel-plugin-transform-class-properties  
yarn add v1.3.2  
[1/4] 🔍 Resolving packages...  
[2/4] 📦 Fetching packages...  
[3/4] 🔗 Linking dependencies...  
[4/4] 🏗 Building fresh packages...  
success Saved 1 new dependency.  
└─ babel-plugin-transform-class-properties@6.24.1  
🌟 Done in 4.71s.
```

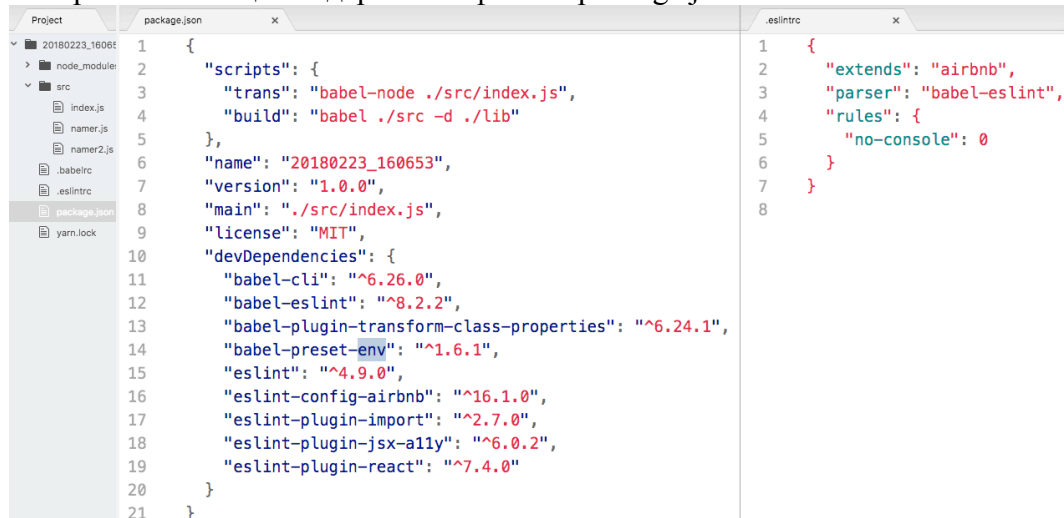
3. Создайте вариант экспортируемого класса без конструктора: namer2.js

```
1  export default class {  
2    first = 'Ilya';  
3    last = 'Goss';  
4  
5    getFullName() {  
6      return `${this.first} ${this.last}`;  
7    }  
8  }
```

4. Осуществите его импорт в файле index.js

```
1  import Namer from './namer2';  
2  
3  const person = new Namer();  
4  console.log(person.getFullName()); // Ilya Goss
```

5. Сверьте с таблицей содержимое файлов `package.json` и `.eslintrc`:



6. Выполните команду

```
npx babel ./src -d ./lib
```

или

```
yarn run build
```

для получения оттранспирированных файлов

7. Убедитесь, что всё выполняется без ошибок:

```
node ./lib/index.js
```

8. Разместите полученный отчёт (лог действий) в репозитории (веб-портфолио).

Тема 2. Серверные программные комплексы на платформе JavaScript и PHP

I. Создание простого веб-сервера на основе Node.js

1. Создайте новый проект: `mkdir $(date +%Y%m%d_%H%M%S) && cd $_ && yarn init -y` или `mkdir $(date +%Y%m%d_%H%M%S) && cd $_ && npm init -y` (<https://kodaktor.ru/g/init>)

2. Установите инструмент nodemon для автоматизации перезапуска сценария и moment для работы с датой и временем: `yarn add --dev nodemon` или `npm i -D nodemon` и `yarn add moment` или `npm i moment`

```
"scripts" : {
  "start": "nodemon"
},
```

3. Установите настройки линтера и создайте нужный файл `.eslintrc`

4. Создайте в папке проекта файл `index.js` с содержимым:

```
1 const http = require('http');
2 const moment = require('moment');
3
4 http.createServer((req, res) => {
5   res.end(moment().format('DD.MM.YYYY HH:mm:ss'));
6 }).listen(4321);
```

5. Запустите сценарий `yarn start` и выполните `curl localhost:4321`

6. Убедитесь, что в консоли отображается текущая дата и время

7. Добавьте к проекту поддержку выдачи данных в формате JSON с выдачей соответствующего заголовка и кодировки UTF-8: <http://kodaktor.ru/git-checkout.gif>

```

1  const http = require('http');
2  const moment = require('moment');
3
4  http.createServer((req, res) => {
5    res.setHeader('Content-Type', 'application/json; charset=utf-8');
6    res.end(JSON.stringify({ date: moment().format('DD.MM.YYYY HH:mm:ss') }));
7  }).listen(4321);

```

8. Перейдите по адресу localhost:4321 в браузере и убедитесь, что выдаётся ответ в формате JSON

9. Осуществите рефакторинг кода так, чтобы коллбэк, отвечающий на запросы, явным образом указывался для события request:

```

1  const http = require('http');
2  const moment = require('moment');
3
4  const server = http.createServer();
5  server.listen(4321);
6  server.on('request', (req, res) => {
7    res.setHeader('Content-Type', 'application/json; charset=utf-8');
8    res.end(JSON.stringify({ date: moment().format('DD.MM.YYYY HH:mm:ss') }));
9  });

```

10. Разместите полученный отчёт (лог действий) в репозитории (веб-портфолио).

(проверочный код: https://github.com/GossJS/node_starters0/tree/event)

II. Моделирование взаимодействия «клиент-сервер» с помощью программы telnet и программы curl

<https://kodaktor.ru/g/telnet> (содержит ссылки на поясняющие видеоролики-скринкасты)

<https://github.com/GossJS/telnet>

1. Установите клиенты telnet и curl или убедитесь, что они установлены на вашем компьютере

2. Подготовьте данные для отправки запросов (ваши URL-кодированные имя и фамилию)

```

> encodeURIComponent('Илья')
'%D0%98%D0%BB%D1%8C%D1%8F'
> encodeURIComponent('Госс')
'%D0%93%D0%BE%D1%81%D1%81'

```

3. Обозначим эти строки как first и second соответственно

Отправьте POST-запрос

на адрес <http://kodaktor.ru/api/req/first/second>

записав в параметр name своё имя

отправьте это значение на проверку преподавателю вместе с тем адресом, на который вы отправили запрос

```
telnet 151.248.115.32 80
POST /api/req/%D0%98%D0%BB%D1%8C%D1%8F/%D0%93%D0%BE%D1%81%D1%81/ HTTP/1.1
Host:kodaktor.ru
Content-Type:application/x-www-form-urlencoded
Content-Length:65
```

2 раза enter

```
name=%D0%98%D0%BB%D1%8C%D1%8F&familyname=%D0%93%D0%BE%D1%81%D1%81
```

1 раз enter

т.е. после Content-Length мы два раза нажимаем Enter, а после name=... один раз.

вы получите в ответ заголовок X-Test со значением вида
POST538c73d64461e13907bb95c51c38bfbc

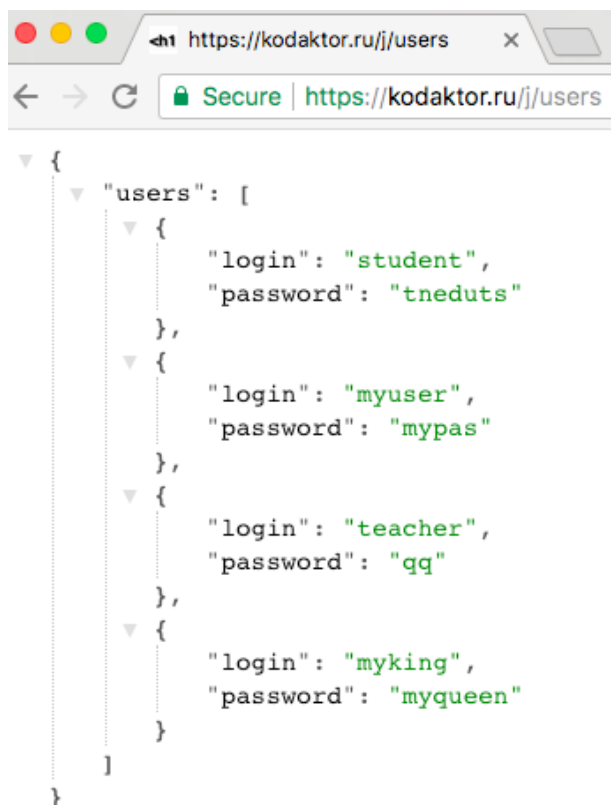
4. отправьте это значение на проверку преподавателю вместе с тем адресом, на который вы отправили запрос

5. Разместите отчёт (лог действий) в репозитории (веб-портфолио).

Тема 3. Решение практических задач обращения с данными на серверной стороне.

I. Создание клиентского веб-приложения для извлечения данных

1. Создайте борд в кодакторе и секцию script в нём
2. Создайте заголовок h1 с текстом «Результат» так, чтобы он был первым потомком элемента body, а также кнопку с надписью «Отправить запрос»
3. Создайте слушатель события «щелчок» к этой кнопке и поместите в него асинхронный код обращения к ресурсу <https://kodaktor.ru/j/users>



4. Необходимо, чтобы асинхронный код извлекал массив, на который ссылается ключ `users` в переменную `users` с помощью деструктуризации и добавлял к текстовому содержимому заголовка значение логина первой записи из массива `users` (т.е. слово `student`).



5. Разместите адрес борда (а также его скриншот и лог действий) в репозитории (веб-портфолио) и в специально созданном форуме в соответствующем курсе в СДО Moodle.

II. Разработка базы данных (БД) типа «ключ-значение» и обращение к ней

1. Создайте аккаунт на сервисе `mlab.com` или создайте инстанс виртуального сервера с `mongo` (для локальных экспериментов можно воспользоваться Docker-контейнером); дальнейшие действия выполняются от имени администратора.
2. Создайте БД: `use j_users`
3. Предоставьте администратору права работы с этой БД `use admin`


```
db.grantRolesToUser( "myUserAdmin", [ { role: "readWrite", db: "j_users" } ] )
```

4. Создайте пользователя внутри этой БД

```
use j_users
db.createUser(
{
  user: "student",
  pwd: "Qwerty.123",
  roles: [
    { role: "readWrite", db: "j_users" }
  ]
}
)
```

5. Добавьте содержимое, идентичное kodaktor.ru/j/users

```
db.users.insertMany([ {"login": "student", "password": "tneduts"}, {"login": "myuser", "password": "mypas"}, {"login": "teacher", "password": "qq"}, {"login": "myking", "password": "myqueen"} ] )
```

6. Проверьте, что по запросу возвращается ожидаемое значение:

```
db.users.findOne({login: "teacher"}).password
qq
```

7. Осуществите подключение к серверу с этой БД из другого места от имени созданного пользователя:

```
mongo mongodb://student:Qwerty.123@151.248.115.32/j_users
```

8. Выполните команду получения списка всех записей

```
db.users.find().map(x=>({username:x.login, password:x.password}))
```

```
[
  {
    "username" : "student",
    "password" : "tneduts"
  },
  {
    "username" : "myuser",
    "password" : "mypas"
  },
  {
    "username" : "teacher",
    "password" : "qq"
  },
  {
    "username" : "myking",
    "password" : "myqueen"
  }
]
```

9. Разместите отчёт о выполненных действиях в веб-портфолио и в специально созданном форуме в соответствующем курсе в СДО Moodle.

Тема 4. Развёртывание и управление веб-ресурсом (веб-портфолио) с помощью клиентского и серверного фреймворка

I. Создание веб-приложения с помощью node.js с использованием серверного шаблонизатора

Цель: создать серверное приложение, которое получает массив объектов, каждый из которых имеет свойство login, из адреса <https://kodaktor.ru/j/users> и использует шаблон pug для вывода в веб-страницу списка этих логинов.

1. Создайте новый проект, инициализируйте зависимости и установите линтер:

```
mkdir $(date +%Y%m%d_%H%M%S) && cd $_ && yarn init -y
bash < (curl -s https://kodaktor.ru/g/eslint_exec)
atom .
```

2. Установите фреймворк express и шаблонизатор pug, а также инструмент axios

3. Установите nodemon

```
yarn add --dev nodemon
```

4. Добавьте в файл package.json раздел scripts с командой start:

```
"scripts" : {
  "start" : "nodemon"
},
"name": "20180311_002511",
"version": "1.0.0",
"main": "./src/index.js",
```

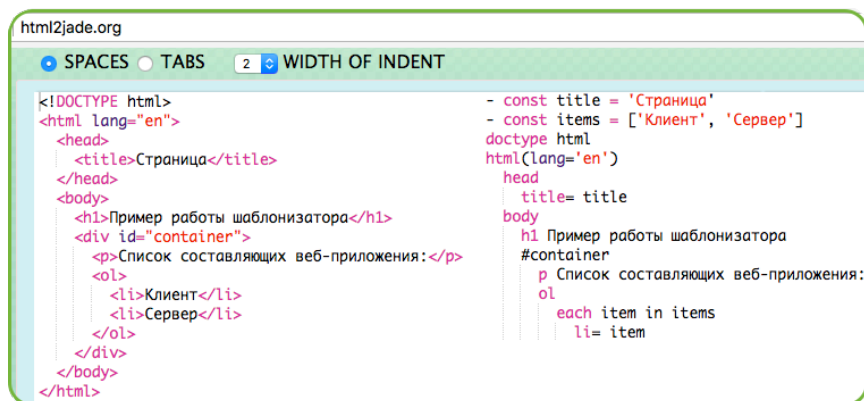
5. Создайте файл index.js в папке ./src

```
1  const express = require('express');
2
3  const PORT = 4321;
4  const app = express();
5  app
6    .get(/hello/, r => r.res.end('Hello world!'))
7
8    .use(r => r.res.status(404).end('Still not here, sorry!'))
9    .use((e, r, res, n) => res.status(500).end(`Error: ${e}`))
10   .set('view engine', 'pug')
11   .listen(process.env.PORT || PORT, () => console.log(process.pid));
```

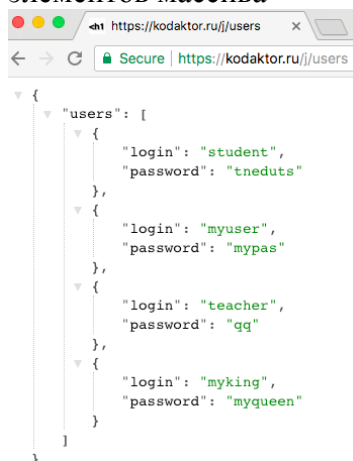
Обратите внимание на подключение шаблонизатора pug (ранее известный как jade)

6. Создайте файл list.pug в папке ./views

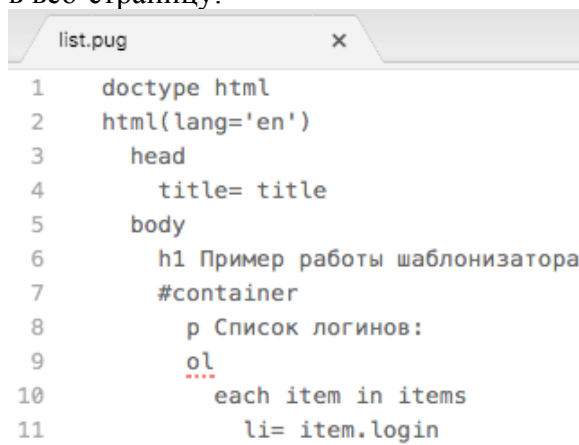
Воспользуйтесь инструментом <http://html2jade.org/> для создания шаблона:



получите с его помощью простой шаблон `list.pug` для итеративной вставки элементов массива



в веб-страницу:



7. Импортируйте метод `get` из зависимости `axios`:

```
const { get } = require('axios');
```

8. Создайте асинхронный обработчик маршрута `users`

9. Используйте метод `render` объекта `Response` для связывания шаблона `list.pug` с данными, полученными из адреса `https://kodaktor.ru/j/users`

```
index.js x
1  const express = require('express');
2  const { get } = require('axios');
3
4  const PORT = 4321;
5  const URL = 'https://kodaktor.ru/j/users';
6  const app = express();
7  app
8    .get(/hello/, r => r.res.end('Hello world!'))
9    .get(/users/, async r => {
10      const { data: { users: items } } = await get(URL);
11      r.res.render('list', { title: 'Список логинов', items });
12    })
13    .use(r => r.res.status(404).end('Still not here, sorry!'))
14    .use((e, r, res, n) => res.status(500).end(`Error: ${e}`))
15    .set('view engine', 'pug')
16    .listen(process.env.PORT || PORT, () => console.log(process.pid));
```

← → ↻ ⓘ localhost:4321/users

Пример работы шаблонизатора

Список логинов:

1. student
2. myuser
3. teacher
4. myking

8. Обеспечьте выполнение требования z7a (<https://kodaktor.ru/z7a>) и разместите работающее приложение так, чтобы оно было доступно для проверки из любого места в Интернете, после чего ознакомьте преподавателя с адресом размещения
9. Разместите отчёт (лог действий) в репозитории (веб-портфолио).

II. Создание веб-приложения с аутентификацией на основе сессий

Цель: создать серверное приложение, которое получает массив объектов, каждый из которых имеет свойство `login`, из адреса `https://kodaktor.ru/j/users` и использует шаблон `pug` для вывода в веб-страницу списка этих логинов при условии ввода верных логина и пароля, которые хранятся по указанному адресу.

Следует взять за основу приложение, созданное в результате выполнения предыдущего задания с шаблонизатором.

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

1. Выполните рефакторинг так, чтобы содержимое документа со списком логинов и паролей было доступно после инициализации приложения:

```
1  const express = require('express');
2  const { get } = require('axios');
3
4  let items; // для последующего заполнения
5  const PORT = 4321;
6  const URL = 'https://kodaktor.ru/j/users';
7  const app = express();
8  app
9    .get(/hello/, r => r.res.end('Hello world!'))
10   .get(/users/, async r => r.res.render('list', { title: 'Список логинов', items }))
11   .use(r => r.res.status(404).end('Still not here, sorry!'))
12   .use((e, r, res, n) => res.status(500).end(`Error: ${e}`))
13   .set('view engine', 'pug')
14   .listen(process.env.PORT || PORT, async () => {
15     console.log(`Старт процесса: ${process.pid}`);
16     // items = (await get(URL)).data.users;
17     ({ data: { users: items } } = await get(URL)); // круглые скобки
18   });
```

- 2.

выполните

```
curl -kSL 'https://kodaktor.ru/g/session_pug' -o './views/login.pug'
```

для получения файла с шаблоном страницы для ввода логина и пароля и добавьте маршрут `/login`

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

3. Выполните

```
yarn add body-parser
```

для установки компонента Express, отвечающего за обработку данных, посланных методом `post`

далее затребуйте соответствующую зависимость в проекте:

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

и добавьте её использование в приложение:

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

после чего добавьте маршрут для обработки POST-запросом

```

3   const bodyParser = require('body-parser');
4
5   let items;
6   const PORT = 4321;
7   const URL = 'https://kodaktor.ru/j/users';
8   const app = express();
9   app
10  .use(bodyParser.json())
11  .use(bodyParser.urlencoded({ extended: true }))
12  .get('/hello/', r => r.res.end('Hello world!'))
13  .get('/login', r => r.res.render('login'))
14  .post('/login/check/', r => {
15    const { body: { login: l } } = r;
16    const user = items.find(({ login }) => login === l);
17    if (user) {
18      if (user.password === r.body.pass) {
19        r.res.send('Good!');
20      } else {
21        r.res.send('Wrong pass!');
22      }
23    } else {
24      r.res.send('No such user!');
25    }
26  })

```

4. Выполните

`yarn add express-session`

для установки компонента Express, отвечающего за обработку сессий

далее затребуйте соответствующую зависимость в проекте:

```
const session = require('express-session');
```

и добавьте её использование в приложение:

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

после добавленной на предыдущем шаге инструкции добавления обработки post-запросов

после чего добавьте запись данных сессии в ветку проверки наличия пользователя, соответствующую успешной проверке совпадения найденного пользователя и его пароля.

```
req.session.auth = 'ok';
```

```

3  const bodyParser = require('body-parser');
4  const session = require('express-session'); ✓
5
6  let items;
7  const PORT = 4321;
8  const URL = 'https://kodaktor.ru/j/users';
9  const app = express();
10 app
11   .use(bodyParser.json())
12   .use(bodyParser.urlencoded({ extended: true }))
13   ✓.use(session({ secret: 'mysecret', resave: true, saveUninitialized: true })))
14   .get('/hello/', r => r.res.end('Hello world!'))
15   .get('/login', r => r.res.render('login'))
16   .post('/login/check/', r => {
17     const { body: { login: l } } = r;
18     const user = items.find(({ login }) => login === l);
19     if (user) {
20       if (user.password === r.body.pass) {
21         r.session.auth = 'ok'; ✓
22         r.res.send('Good!');
23       } else {
24         r.res.send('Wrong pass!');
25       }
26     } else {
27       r.res.send('No such user!');
28     }
29   })

```

5. Добавьте функцию `checkAuth` промежуточного программного обеспечения (конвейера, `middleware`) для обработки защищённых маршрутов и защите маршрута `/users` ею:

```

10 const checkAuth = (r, res, next) => {
11   if (r.session.auth === 'ok') {
12     next();
13   } else {
14     res.redirect('/login');
15   }
16 };
17 app
18   .use(bodyParser.json())
19   .use(bodyParser.urlencoded({ extended: true }))
20   .use(session({ secret: 'mysecret', resave: true, saveUninitialized: true })))
21   .get('/hello/', r => r.res.end('Hello world!'))
22   .get('/login', r => r.res.render('login'))
23   .post('/login/check/', r => {
24     const { body: { login: l } } = r;
25     const user = items.find(({ login }) => login === l);
26     if (user) {
27       if (user.password === r.body.pass) {
28         r.session.auth = 'ok';
29         r.res.send('Good!');
30       } else {
31         r.res.send('Wrong pass!');
32       }
33     } else {
34       r.res.send('No such user!');
35     }
36   })
37   ✓
38   .get('/users/', checkAuth, async r => r.res.render('list', { title: 'Список логинов', items }))

```

6. Добавьте маршрут `/logout` для прекращения сессии и убедитесь, что попытка посетить маршрут `/users` без предварительного залогинивания приводит к перемещению к маршруту `/login`, после успешного заполнения формы по которому верными логином и паролем повторное посещение

маршрута `/users` будет уже успешным, и так до посещения маршрута `/logout` или до окончания срока сессии.

7. Реализуйте требования z7a (маршрут `/author`, выдающий в верной кодировке ваши имя и фамилию), реализуйте запоминание посещённого маршрута (до авторизации) так, чтобы после переадресации на форму и успешного логина приложение возвращалось к затребованному ранее закрытому маршруту, обеспечьте работу приложения в открытом доступе для проверки и разместите отчёт (лог действий) в репозитории (веб-портфолио).

Проверочный код: https://github.com/GossJS/express_starters0/tree/sess6
см. также https://kodaktor.ru/g/lr_sessions

2. Типовые задания для инвариантной самостоятельной работы по темам

Задания для самостоятельной инвариантной работы обобщают и расширяют задания лабораторных работ, преимущественно предполагают поисковую деятельность в аспекте модификация и совершенствования алгоритмов и моделей, предложенных в лабораторных работах и лекционном материале по теме. Эти задания нацелены на формирование видов деятельности, которые обучающиеся должны уметь осуществлять. Задания представлены в СДО Moodle в электронном учебном курсе по дисциплине «Компьютерный практикум»

Выполнение задания предполагает следующие виды деятельности:

- поиск информации в Интернете по изучаемым технологиям и инструментам, анализ справочных материалов и документации,
- развёртывание и настройку окружения (программной среды, IDE, папки проекта, репозитория);
- проектирование формата данных веб-приложения или формального языка (языка разметки), основанного на XML;
- разработку сценария или приложения на веб-языке (языке программирования высокого уровня в области веб-ресурсов):
 - JavaScript;
 - и, возможно, другом языке по выбору обучающихся;
- выявление и исправление синтаксических ошибок, выполнение транспиляции, тестирования работы сценария.
- составление отчета о выполненном задании в виде слайдов и/или онлайн-документации, размещаемой в Git-репозитории как части веб-портфолио обучающегося.

Критерии оценивания. Задание считается выполненным, если сценарий или отчёт соответствует заданию, сценарий успешно проходит процедуру модульного тестирования и сопровождается репозиторием.

№ темы	Содержание самостоятельной работы обучающихся
1	<ol style="list-style-type: none">1. Транспилиция и сборка проекта с использованием директив импорта и экспорта в код с модульностью CommonJS2. Транспилиция и сборка проекта с использованием оператора связывания или иных трансформационных плагинов транспилятора3. Формирование отчета по выполнению задания (тема 1) и размещение его в портфолио.
2	<ol style="list-style-type: none">1. Разработка приложения на основе шаблона Node.js HTTP Simple Server, выдающего результаты вычислений в формате JSON2. Эмулирование отправки HTTP-запроса серверу методом GET с помощью клиентов telnet с записью скринкаста совершаемых действий.

	<p>3. Отправка HTTP-запроса серверу методом GET и POST с помощью приложения сURL с записью скринкаста совершаемых действий.</p> <p>4. Формирование отчета по выполнению задания (тема 2) и размещение его в портфолио.</p>
3	<p>1. Разработка приложения, извлекающего данные из файла в формате JSON</p> <p>2. Проектирование веб-базы данных и разработка приложения на основе фреймворка Express с реализацией маршрутизации и операций CRUD над простой БД с обработкой запросов REST</p> <p>3. Формирование отчета по выполнению задания (тема 3) и размещение его в портфолио.</p>
4	<p>1. Подключение серверного шаблонизатора к веб-приложению и формирование основной структуры веб-портфолио.</p> <p>2. Анализ и настройка серверной компоненты веб-сайта. Анализ и настройка механизма аутентификации веб-сайта в том числе на основе механизма сессий;</p> <p>3. Формирование отчета по выполнению задания (тема 4) и размещение его в веб-портфолио.</p>

3. Типовые задания для вариативной самостоятельной работы по темам

Задания для вариативной самостоятельной работы позволяют углубить и детализировать важные аспекты содержания дисциплины, представляющие интерес для каждого конкретного обучающегося, включая углубление и детализацию теоретической подготовки и формирование исследовательской деятельности, для реализации чего предназначены задания на подготовку выступлений, докладов и аналитики.

Для выполнения этих заданий, например, проектирования дизайна главной страницы мобильного приложения, требуется выполнение следующих действий:

- анализ программных средств и инструментов, их документации;
- прототипирование (моделирование) веб-ресурса;
- анализ и использование фреймворков, логирование

При подготовке выступления следует руководствоваться следующей дорожной картой презентации:

- обзор по теме (сравнение, таблица, ... - слайды в google drive или инструменте вещания слайдов);
- демонстрация в live-режиме (slides.com, например <http://slides.com/elizabethanatskaya-1/deck-2#/12> и др.);
- выводы;
- примеры заданий для аудитории на овладение материалом (интерактив);
- поддержка в репозитории (ссылки на слайды / ресурсы / ...).

Критерии оценивания. Задание считается выполненным, если сценарий или отчет соответствует заданию, сценарий успешно проходит процедуру модульного тестирования и сопровождается репозиторием, выступление логично и последовательно охватывает заявленную тему.

№ темы	Содержание самостоятельной работы обучающихся
1	1. Сборка проекта с помощью бандлера Webpack 2. Сборка проекта с помощью бандлера Rollup 3. Сборка проекта с использованием нативных модулей на базе элементов script type="module"
2	1. Разработка REST-микросервиса с аутентификацией на базе сессий 2. Разработка REST-микросервиса с аутентификацией на базе технологии «паспорт» 3. Разработка REST-микросервиса с аутентификацией на базе сессий на основе

	<p>PHP с проверкой корректности работы сессий</p> <p>4. Разработка серверной компоненты приложения удалённого доступа к базам данных и экспериментальная проверка корректности её работы</p>
3	<p>1. Подготовка презентации по Mongo и Mongoose</p> <p>2. Подготовка материалов для выступления по NoSQL-решениям</p> <p>3. Подготовка материалов для выступления по сравнению MySQL и NoSQL-решений</p>
4	<p>1. Развёртывание веб-ресурса на платформе Heroku</p> <p>2. Развёртывание веб-ресурса на платформе Digital Ocean</p> <p>3. Развёртывание веб-ресурса на платформе vscale.io с использованием Docker</p>