



**Министерство науки и высшего образования Российской
Федерации Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский государственный технический
университет имени Н.Э. Баумана
(национальный исследовательский
университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №7

По курсу «Разработка интернет приложений»

Подготовила:

Студентка группы ИУ5-55Б.

Очеретная С.В.

18.12.2020

Проверил:

Преподаватель кафедры ИУ5

Гапанюк Ю.Е.

Москва, 2021 г.

Цель лабораторной работы: изучение возможностей создания пользовательского интерфейса в веб-приложениях.

Задание: разработать пользовательский интерфейс для работы с REST API. Использовать REST API, разработанный в 6 лабораторной работе.

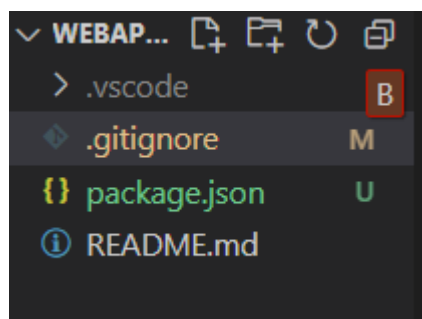
Ход работы

Создадим проект прописав `npm init`

Получаем созданный `package.json`

```
package.json > ...
{
  "name": "lab7",
  "version": "1.0.0",
  "description": "Репозиторий для курса \"Разработка Интернет приложений\" <br/>\r Студентка: Очеретная Светлана <br/>\r Группа: ИУ5-55Б",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/svetlanlka/webappdevelopment.git"
  },
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/svetlanlka/webappdevelopment/issues"
  },
  "homepage": "https://github.com/svetlanlka/webappdevelopment#readme"
}
```

Добавим `gitignore`



Мы создали проект, теперь давайте начнем его писать. Когда пользователь заходит на сайт, то ему сначала подгружается `index.html` файл с базовой версткой, а потом уже подгружаются стили и скрипты.

- Создаем файл `index.html`
- Чтобы отдавать этот файл, напишем свой сервер, для этого скачаем `express` через команду `npm install express --save-dev`
У нас добавился пакет в `package.json` и добавилась папка `node_modules`, где хранится наш пакет
- Мы установили `express`, теперь нам нужно настроить наш сервер. Для этого создадим `server/server.js`

```
'use strict';

const express = require('express');
const path = require('path');
const app = express();
const htmlPath = path.resolve(__dirname, '..', 'public/index.html');
const staticPath = path.resolve(__dirname, '..', 'src');

const port = 8080;
const ip = 'http://localhost';

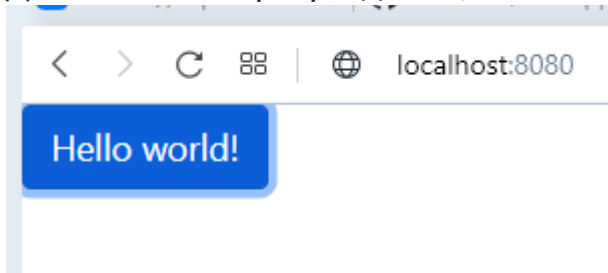
app.use('/', express.static(staticPath));

app.all('*', (req, res) => {
  const path_ = req.path === '/' ? 'index.html' : req.path
  res.contentType(path.basename(path_));
  console.log(req.path)
  res.sendFile(path.resolve(__dirname, '..', 'public/' + path_));
});

app.listen(port, () => {
  console.log(`listening at ${ip}:${port}`);
});
```

- Для запуска сервера используем команду `node server/server.js`. Теперь если мы перейдем на по урлу <http://localhost:8080>, то сможем увидеть наш сайт
- Добавим bootstrap для верстки

Добавим кнопку и убедимся, что bootstrap подключен:



Простая страничка на JS

Теперь попробуем нарисовать нашу кнопку с помощью JS. Для того, чтобы в JS получить доступ к нашему html у нас должен быть какой-то корневой элемент, который будет нашим родителем и к которому мы будем добавлять остальные элементы.

- Добавим корневой элемент в `index.html`

Теперь у нас есть корневой элемент, к которому мы можем обратиться из JS. Создадим js, подключим его и попробуем обратиться к html.

- Создаем файл main.js. Для доступа к html мы будем использовать **getElementById**

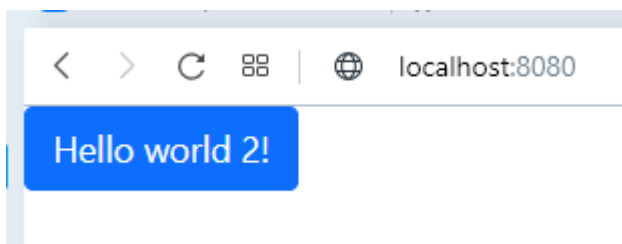
```
const root = document.getElementById('root');
```

- Подключаем этот файл в index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Donuts</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css">
</head>
<body>
  <div id="root"></div>
  <script src="main.js" type="module"></script>
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.bundle.min.
js"></script>
</body>
</html>
```

- Попробуем добавить кнопку:

```
root.insertAdjacentHTML('beforeend', '<button type="button" class="btn btn-
primary">Hello world 2!</button>')
```



Структурирование проекта

Добавим папки:

- public/pages - тут будут лежать наши страницы
- public/components - тут будут лежать наши компоненты
- public/modules - тут будут лежать наши утилиты, которые нам позже пригодятся

Теперь попробуем переписать нашу страницу под нашу новую архитектуру

```
c > JS main.js > ...
import MainPage from "../pages/main/MainPage.js";

const root = document.getElementById('root');

const mainPage = new MainPage(root);
mainPage.render();
```

< > ↺ ☐ | 🌐 localhost:8081

Hello world 3!

Вынесем кнопку в компонент и добавим ее на странице.

```
export default class ButtonComponent {
  constructor(root) {
    this.root = root;
  }

  render(props, onClick) {
    const btnWrapper = document.createElement('div');
    btnWrapper.innerHTML = `<button type="button" class="btn btn-primary">${props.sign}</button>`;
    const btn = btnWrapper.firstChild;
    btn.addEventListener('click', onClick);
    this.root.insertAdjacentElement('beforeend', btn);
  }
}
```

```
render() {
  const button = new ButtonComponent(this.root)
  const props = {sign: "Hello world 4!"}
  button.render(props)
```

< > ↺ ☐ | 🌐 localhost:8081

Hello world 4!

Верстка главной страницы под наше API

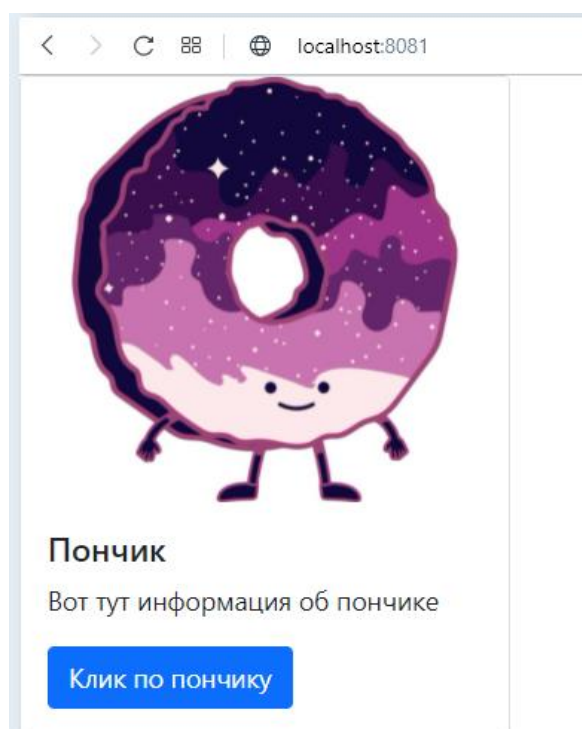
Теперь модифицируем нашу страницу так, чтобы она была готова для отображения данных, которые отдает наше API. Для отображения одного пончика будем использовать карточки

```
export default class DonutCardComponent {
  constructor(root) {
    this.root = root;
  }

  getHTML() {
    return `
      <div class="card" style="width: 300px;">
        
        <div class="card-body">
          <h5 class="card-title">Пончик</h5>
          <p class="card-text">Вот тут информация об пончике</p>
          <button class="btn btn-primary">Клик по пончику</button>
        </div>
      </div>
    `;
  }

  render() {
    const html = this.getHTML();
    this.root.insertAdjacentHTML('beforeend', html);
  }
}
```

```
render() {
  const donutCard = new DonutCardComponent(this.root);
  donutCard.render();
}
```



Отлично, у нас все работает, но нам бы хотелось прокидывать данные в компонент, а то сейчас у нас данные захардкожены в компоненте.

- Добавим отрисовку компонента из данных

```
getHTML(props) {
  return (
    <div class="card card-donut">
      
      <div class="card-body">
        <h2 class="card-title">${props.name}</h2>
        <h3 class="card-text">${props.cost} руб.</h3>
        <p class="card-text">${props.info}</p>
        <button class="btn btn-primary" id="click-card-${props.id}" data-id="${props.id}">Клик по карте пончика</button>
      </div>
    </div>
  )
}

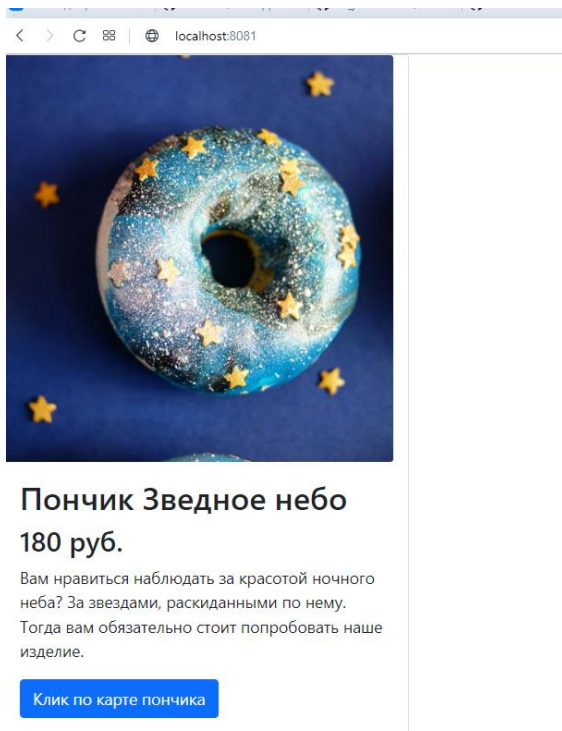
render(props) {
  const html = this.getHTML(props)
  this.root.insertAdjacentHTML('beforeend', html)
}
```

- Теперь нам нужно прокидывать данные в компонент со страницы. Функция `getData` будет выдавать тестовые данные, потом в ней мы будем делать поход на бекенд и получать реальные данные

```
getData() {
  return {
    id: 1,
    picture: "uploads/donut-1.jpg",
    name: "Пончик Звездное небо",
    cost: 180,
    info: "Вам нравится наблюдать за красотой ночного неба? За звездами, раскиданными \ по нему. Тогда вам обязательно стоит попробовать наше изделие."
  }
}

render() {
  const data = this.getData()
  const donutCard = new DonutCardComponent(this.root)
  donutCard.render(data)
}
```

Полученная карта пончика:



Теперь отрисуем список пончиков.

- Добавим родительский элемент на главной странице, куда будем вставлять все наши компоненты

```
get page() {
  return document.getElementById('main-page')
}

getHTML() {
  return (
    <div id="main-page" class="d-flex flex-wrap"><div/>
  )
}

render() {
  const data = this.getData()
  const donutCard = new DonutCardComponent(this.root)
  donutCard.render(data)

  this.parent.innerHTML = ''
  const html = this.getHTML()
  this.parent.insertAdjacentHTML('beforeend', html)
```


- Теперь поменяем функцию `getData`, чтобы она отдавала массив данных и отрисовываем компоненты по этому массиву

```
getData() {  
  return [{  
    id: 1,  
    picture: "uploads/donut-1.jpg",  
    name: "Пончик Звездное небо",  
    cost: 180,  
    info: "Вам нравится наблюдать за красотой ночного неба? За звездами, раскиданными \\  
по нему. Тогда вам обязательно стоит попробовать наше изделие."  
  },  
  {  
    id: 2,  
    picture: "uploads/donut-2.jpg",  
    name: `Пончик "Космическая радость"`,  
    cost: 130,  
    info: "Этот пончик космически красив и обязательно порадует вас своим вкусом. "  
  },  
  {  
    id: 3,  
    picture: "uploads/donut-3.jpg",  
    name: `Пончик "Галактика"`,  
    cost: 145,  
    info: "Вкус этого пончика превосходен. Говорят, что его создатели обладали \\  
знаниями, присланными издалека, с конца нашей галактики."  
  },  
]  
}
```

```
render() {  
  const data = this.getData()  
  data.forEach((item) => {  
    const donutCard = new DonutCardComponent(this.page)  
    donutCard.render(item)  
  })  
  
  this.parent.innerHTML = ''  
  const html = this.getHTML()  
  this.parent.insertAdjacentHTML('beforeend', html)
```

Отлично, мы смогли отрисовать сразу несколько компонентов, но если мы попробуем нажать на кнопку, то ничего не произойдет. Добавим обработку нажатия на кнопку. Для этого нам нужно внутри компонента подписаться на событие клик по кнопке и обработать его. Функцию, которая будет срабатывать по клику будем прокидывать в компонент из страницы

- Добавим нашей кнопке уникальный id, чтобы по нему мы могли найти кнопку и подписаться на событие клика

```
<button class="btn btn-primary" id="click-card-${props.id}" data-id="${props.id}">Клик по карте пончика</button>
</div>
```

- Теперь у кнопки есть уникальный id, и мы можем подписаться на клик на нее. Для подписки на событие используется функция addEventListener

```
addListeners(data, listener) {
  document
    .getElementById(`click-card-${data.id}`)
    .addEventListener("click", listener);
}

render(props, listener) {
  const html = this.getHTML(props)
  this.root.insertAdjacentHTML('beforeend', html)
  this.addListeners(props, listener);
}
```




- Теперь добавим на главной функцию, которая будет срабатывать по нажатию и прокинем ее в компонент. При создании кнопки мы добавили ей data атрибут, чтобы при обработке мы могли его достать и узнать по какому элементу кликнули.

```
clickCard(e) {
  const cardId = e.target.dataset.id
}

render() {
  this.root.innerHTML = ''
  const html = this.getHTML()
  this.root.insertAdjacentHTML('beforeend', html)

  const data = this.getData()
  data.forEach((item) => {
    console.log("page: ", this.page)
    const donutCard = new DonutCardComponent(this.page)
    donutCard.render(item, this.clickCard.bind(this))
  })
}
```

Итог:

		
<p>Пончик Звездное небо 180 руб.</p> <p>Вам нравится наблюдать за красотой ночного неба? За звездами, раскиданными по нему. Тогда вам обязательно стоит попробовать наше изделие.</p> <p>Клик по карте пончика</p>	<p>Пончик "Космическая радость" 130 руб.</p> <p>Этот пончик космически красив и обязательно порадует вас своим вкусом.</p> <p>Клик по карте пончика</p>	<p>Пончик "Галактика" 145 руб.</p> <p>Вкус этого пончика превосходен. Говорят, что его создатели обладали знаниями, присланными издалека, с конца нашей галактики.</p> <p>Клик по карте пончика</p>

Теперь сверстаем страницу пончика под наше API

- Создаем страницу пончика `public/donut/DonutPage.js`. Наша страница будет принимать дополнительный аргумент `id` выбранного пончика

```
export default class DonutPage {
  constructor(root, id) {
    this.root = root;
    this.id = id;
  }

  getData() {
    return {
      id: 3,
      picture: "uploads/donut-3.jpg",
      name: `Пончик "Галактика"`,
      cost: 145,
      info: "Вкус этого пончика превосходен. Говорят, что его создатели обладали \
знаниями, присланными издалека, с конца нашей галактики."
    };
  }

  get page() {
    return document.getElementById('donut-page');
  }

  getHTML() {
    return (
      `
      <div id="donut-page"></div>
      `
    );
  }

  render() {
    this.root.innerHTML = '';
    const html = this.getHTML();
    this.root.insertAdjacentHTML('beforeend', html);
  }
}
```

- Теперь создадим компонент, который будет отрисовываться на странице пончика `public/components/donut/donutComponent.js`

```
export default class DonutComponent {
  constructor(root) {
    this.root = root;
  }

  getHTML(props) {
    return (
      `
      <div class="card card-donut">
        
        <div class="card-body">
          <h2 class="card-title">${props.name}</h2>
          <h3 class="card-text">${props.cost} py6.</h3>
          <p class="card-text">${props.info}</p>
          <button class="btn btn-primary" id="click-card-${props.id}" data-id="${props.id}">Клик по карте пончика</button>
        </div>
      </div>
      `
    );
  }

  render(props) {
    const html = this.getHTML(props);
    this.root.insertAdjacentHTML('beforeend', html);
  }
}
```

- Добавим отрисовку компонента на странице пончика

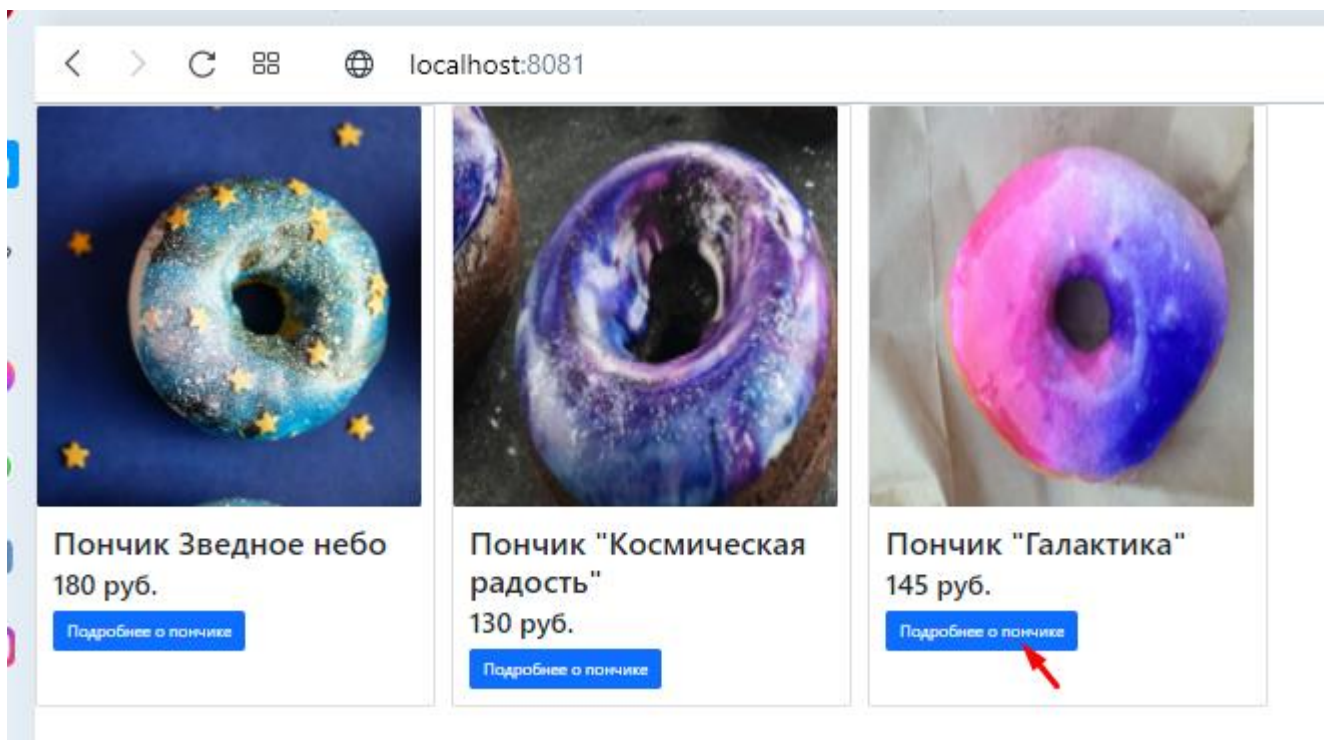
```
const data = this.getData()
const donut = new DonutComponent(this.page)
donut.render(data)
```

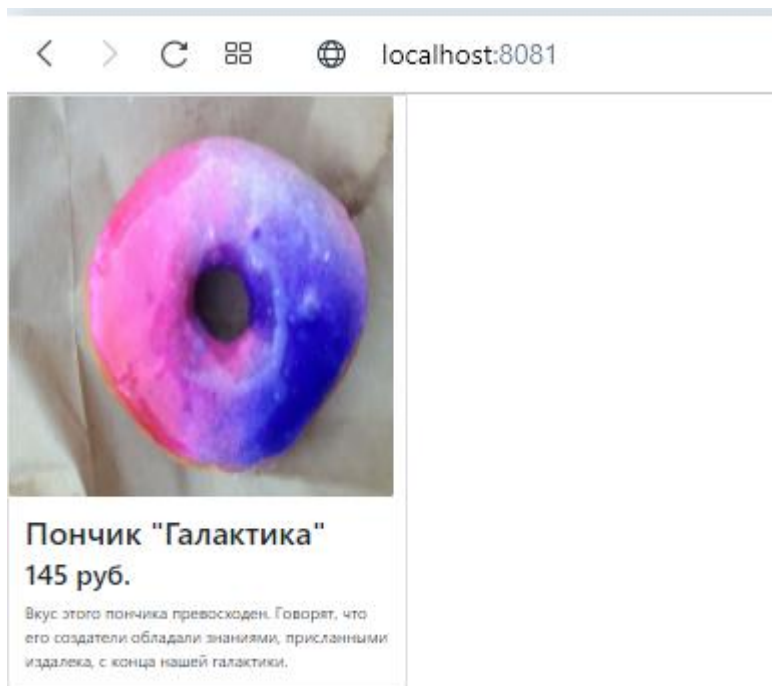
- Теперь у нас есть страница пончика, сделаем так, чтобы при нажатии на карточку на главной странице эта страница открывалась

```
clickCard(e) {
  const cardId = e.target.dataset.id

  const donutPage = new DonutPage(this.root, cardId)
  donutPage.render()
}
```

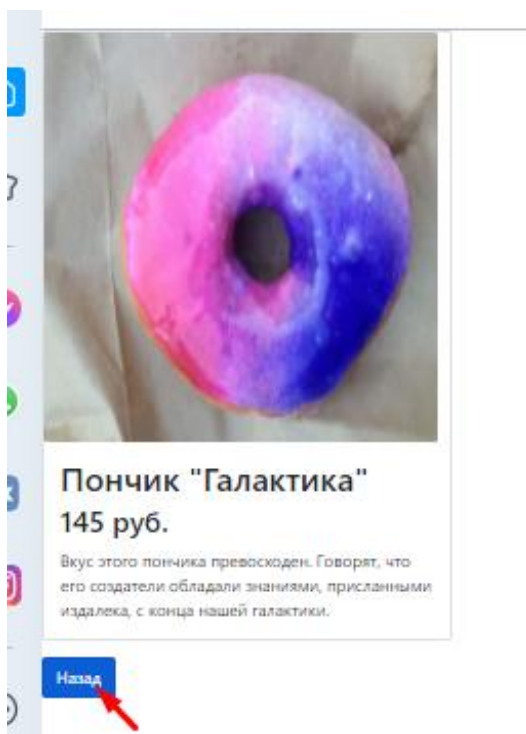
Итог:





Добавим кнопку, которая будет возвращать на главную страницу

```
const backBtn = new ButtonComponent(this.page);
backBtn.render({sign: 'Назад'}, (e) => {
  const mainPage = new MainPage(this.root);
  mainPage.render();
});
```



Подключение API

У нас есть приложение, но нам нужно его соединить с нашим бекендом, чтобы оно получало данные. Для этого мы будем делать fetch запросы

- Создадим модуль для работы с бекендом `public/modules/ajax.js`. В этом модуле у нас есть функция `get` для выполнения запроса на бекенд. После этого полученные данные парсятся и нам отдается объект из статуса и данных

```
class Ajax {
  async get(url) {
    const response = await fetch(url, {
      method: "GET"
    });
    const responseData = await response.json();

    return {
      status: response.status,
      data: responseData
    };
  }
}

export const ajax = new Ajax();
```

- Так же для удобства создадим модуль, где будут храниться наши урлы, чтобы они были прописаны в одном месте. Создаем `public/modules/urls.js`

```
class Urls {
  constructor() {
    this.url = 'http://localhost:8080/';
  }

  donuts() {
    return `donuts/`;
  }

  donut(id) {
    return `donuts/${id}/`;
  }
}

const urls = new Urls();

export default urls;
```

- Теперь заменим нашу функцию `getData` на главной странице, чтобы она работала с нашим модулем

```

getData() {
  return ajax.get({url: urls.donuts()});
}

```

```

this.getData().then(({response}) => {
  response.forEach(item => {
    const donutCard = new DonutCardComponent(this.page)
    donutCard.render(item, this.clickCard.bind(this));
  });
});
}

```

- Заменим нашу функцию getData на странице пончика

```

getData() {
  return ajax.get({url: urls.donut(this.id)});
}

```

```

const donut = new DonutComponent(this.page);
this.getData().then(({response}) => {
  console.log("data:", response)
  donut.render(response);

  const backButton = new ButtonComponent(this.page);
  backButton.render({sign: 'Назад'}, (e) => {
    const mainPage = new MainPage(this.root);
    mainPage.render();
  });
});

```

В методичке оказался не совсем рабочий код и файл ajax.js был переписан.

<https://github.com/Svetlanlka/WebAppDevelopment/blob/lab7/public/module/ajax.js>

Перед запуском лабораторной следует правильно настроить settings предыдущей 6 лабораторной, которая раздает данные нашей БД, созданной в 5 лабораторной

В 6 лабе в settings.py добавим следующие настройки


```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework', # REST Framework
    "corsheaders", # CORS
    'stocks',
    'bootstrap4',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    "corsheaders.middleware.CorsMiddleware",
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

#CORS
CORS_ALLOWED_ORIGINS = [
    "http://localhost:8081",
    "http://127.0.0.1:8081",
    "http://localhost:3000",
    "http://127.0.0.1:3000",
]

CORS_ALLOW_CREDENTIALS = True

```

В fetch-запрос необходимо добавить mode: 'cors' и credentials: 'include'

```

const fetchParams = {
  body: JSON.stringify(requestParams.body),
  mode: 'cors',
  credentials: 'include',
  headers: {
    'Content-Type': 'application/json',
  },
  method: requestParams.method,
};

```