МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительноӗ техники и автоматизированных систем

Расчётно-графическое задание  
по дисциплине: Технологии Web-программирования

Выполнил: ст. гр. ПВ-41

Мишина Ю. А.

Проверил: Картамышев С. В.

Белгород 2020г.

**Оглавление**

[Постановка задачи 4](#_Toc61388474)

[HTML. Разработка макетов и верстка шаблонов web-приложения с помощью языков HTML и CSS 5](#_Toc61388475)

[Выполнение 5](#_Toc61388476)

[Разработка 6](#_Toc61388477)

[Полученный результат 10](#_Toc61388478)

[Весь код можно посмотреть по ссылке: 12](#_Toc61388479)

[Клиентское программирование. React JS 13](#_Toc61388480)

[Выполнение 13](#_Toc61388481)

[Главный компонент 14](#_Toc61388482)

[Пример кода компонента Header сайта 14](#_Toc61388483)

[Запуск приложения 15](#_Toc61388484)

[Весь код можно посмотреть по ссылке: 15](#_Toc61388485)

[Серверное программирование. Spring Boot 16](#_Toc61388486)

[Выполнение 16](#_Toc61388487)

[Создаём точку запуска нашего приложения 16](#_Toc61388488)

[Создаём контроллеры 17](#_Toc61388489)

[Добавление приложения в Docker 17](#_Toc61388490)

[Результат 19](#_Toc61388491)

[Весь код можно посмотреть по ссылке: 19](#_Toc61388492)

[Разработка и проектирование базы данных 20](#_Toc61388493)

[Выполнение 20](#_Toc61388494)

[Подключение необходимых библиотек 20](#_Toc61388495)

[Структура базы данных 21](#_Toc61388496)

[Создание сущностей 21](#_Toc61388497)

[Создание JPA репозиториев 22](#_Toc61388498)

[Работа с PostgeSQL 22](#_Toc61388499)

[Демонстрация работы 23](#_Toc61388500)

[Результат 23](#_Toc61388501)

[Весь код можно посмотреть по ссылке 23](#_Toc61388502)

[REST API 24](#_Toc61388503)

[Выполнение 24](#_Toc61388504)

[Разработка DTO 24](#_Toc61388505)

[Разработка контроллеров 24](#_Toc61388506)

[Разработка мапперов 25](#_Toc61388507)

[Разработка сервисов 25](#_Toc61388508)

[Пример работы с REST API 26](#_Toc61388509)

[Весь код можно посмотреть по ссылке 28](#_Toc61388510)

[Работа с HTTP запросами 29](#_Toc61388511)

[Выполнение 29](#_Toc61388512)

[Back 29](#_Toc61388513)

[Front 33](#_Toc61388514)

[Возможные проблемы 35](#_Toc61388515)

[Решение проблемы 35](#_Toc61388516)

[Состояние базы данных 36](#_Toc61388517)

[Результаты выполнения GET-запросов 37](#_Toc61388518)

[Общий вид получившегося сайта 38](#_Toc61388519)

[Весь код можно посмотреть по ссылке: 40](#_Toc61388520)

[Вывод 41](#_Toc61388521)

# **Постановка задачи**

В качестве разрабатываемого проекта была выбрана тематика новостного сайта об играх. Сайт представляет из себя главную страницу со списком новостей в виде карточек, возможность перейти на новость с ее детальным просмотром и посмотреть информацию о сайте.

Стек разработки:

Для разработки фронта были использованы технологии: React JS, HTML5, CSS3

Для разработки бека были использованы технологии: Maven, Spring, Spring Boot, Junit и многое другое.

Для настройки системы контроля версий и автоматизированной сборки: Git, GitHab, Docker, DockerCompose.

# **HTML. Разработка макетов и верстка шаблонов web-приложения с помощью языков HTML и CSS**

**Цель**: в данном пункте нем необходимо, используя HTML + CSS + BootStrap наверстать тестовый вариант нашего будущего сайта.

## **Выполнение**

**HTML** (HyperText Markup Language - язык гипертекстовой разметки) не является языком программирования, это язык разметки, используемый для определения структуры веб-страниц, посещаемых пользователями. Они могут иметь сложную или простую структуру, всё зависит от замысла и желания веб-разработчика. HTML состоит из ряда элементов, которые можно использовать для того, чтобы обернуть или разметить различные части информации, чтобы это все имело определенный вид или срабатывало определенным способом. Встроенные, так называемые, теги могут преобразовать часть содержимого в гиперссылку, по которой можно перейти на другую веб-страницу, выделить курсивом слова и тому подобное.

Основными частями HTML-элемента являются:

- открывающий тег (он состоит из названия элемента (в данном случае, p), помещенного внутри угловых скобок, данный тег служит признаком начала элемента, с этого момента тег начинает влиять на следующее после него содержимое)

- закрывающий тег (выглядит как и открывающий, но содержит слэш перед названием тега, он служит признаком конца элемента)

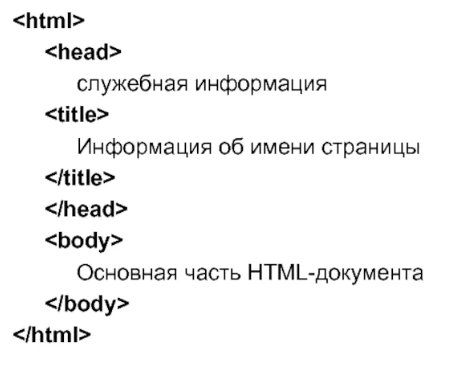
- содержимое (в данном случае содержимым является простой текст)

- элемент: открывающий тег + закрывающий тег + содержимое.

Изначально язык HTML был задуман и создан как средство структурирования и форматирования документов без их привязки к средствам воспроизведения (отображения). В идеале, текст с разметкой HTML должен был без стилистических и структурных искажений воспроизводиться на оборудовании с различной технической оснащенностью (цветной экран современного компьютера, монохромный экран органайзера, ограниченный по размерам экран мобильного телефона или устройства и программы голосового воспроизведения текстов). Однако современное применение HTML очень далеко от его изначальной задачи. Например, тег

|  |
| --- |
| предназначен для создания в документах таблиц, но иногда используется и для оформления размещения элементов на странице. |

Структура HTML-файла:



Тег <html> определяет начало HTML-файла, внутри него хранится заголовок (<head>) и тело документа (<body>).

Заголовок документа, как еще называют блок <head>, может содержать текст и теги, но содержимое этого раздела не показывается напрямую на странице, за исключением контейнера <title>. Данный тег необходим для задания управляющей информации веб-страницы.

Тело документа <body> предназначено для размещения содержательной части веб-страницы.

**CSS** (Cascading Style Sheets — каскадные таблицы стилей) — формальный язык описания внешнего вида документа, написанного с использованием языка разметки. Преимущественно используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML, но может также применяться к любым XML-документам. CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось отделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана).

Структура CSS-документа представляет из себя набор правил, где каждое правило CSS имеет две основные части — селектор и блок объявлений. Селектор, расположенный в левой части правила до знака «{» определяет, на какие части документа (возможно, специально обозначенные) распространяется правило. Блок объявлений располагается в правой части правила. Он помещается в фигурные скобки, и, в свою очередь, состоит из одного или более объявлений, разделенных знаком «;». Каждое объявление представляет собой сочетание свойства CSS и значения, разделенных знаком «:». Селекторы могут группироваться в одной строке через запятую. В таком случае свойство применяется к каждому из них.

### **Разработка**

Подключаем все необходимые стили и библиотеки. После чего начинам верстать нашу основную часть.

Например, таким образом будет выглядеть код главной страницы сайта.

<body>  
<div class="site">  
 <div class="header">  
 <div class="header\_top">  
 <div class="top\_left">  
 <div class="left\_logo">  
 <a href="" class="logo">  
 <img src="img/snail.png" alt="">  
 </a>  
 </div>  
 <div class="left\_title">  
 <div class="title\_in">  
 <a href="" class="in\_link">SNAIL</a>  
 </div>  
 <div class="title\_description">Новости из мира компьютерных и мобильных игр</div>  
 </div>  
 </div>  
 <div class="top\_right">  
 <div class="right\_addres">  
 <div class="addres\_ico"></div>  
 <div class="addres\_in">  
 <div class="in\_title">Разместить рекламу</div>  
 <div class="in\_description"></div>  
 <div class="in\_number">  
 <a href="" class="number\_cool">8 (000) 000-00-00</a>  
 <a href="" class="number\_cool">8 (000) 000-00-00</a>  
 </div>  
 </div>  
 </div>  
 <div class="right\_button">  
 <div class="button\_left">  
 <div class="left\_box">  
 <a href="login.html" class="left\_title">ВОЙТИ</a>  
 <div class="title\_ico"></div>  
 </div>  
 </div>  
 </div>  
  
 </div>  
 </div>  
  
 <div class="header\_menu">  
 <ul class="menu\_in">  
 <li class="in\_after">  
 <a href="" class="after\_in">Главная</a>  
 <a href="info.html" class="after\_in">О нас</a>  
 </li>  
 </ul>  
 </div>  
 </div>  
  
 <div class="content">  
 <div class="content\_in">  
  
 <div class="in\_box">  
 <div class="box\_top">  
 <div class="top\_img">  
 <a href="" class="img\_link">  
 <img src="https://i.pinimg.com/originals/57/eb/a9/57eba9d157d5a5f497ba85c9b1f8aa8b.jpg" alt="">  
 </a>  
 </div>  
 <div class="top\_title">  
 <a href="" class="title\_link">ГЕЙМПЛЕЙ WATCH DOGS: LEGION НА XBOX SERIES X С ТРАССИРОВКОЙ ЛУЧЕЙ</a>  
 </div>  
 </div>  
 <div class="box\_bottom">  
 <div class="bottom\_description">Watch Dogs: Legion уже доступна на PC и приставках текущего поколения, однако приверженцам консольного гейминга, собирающимся перейти на некстген, есть смысл дождаться соответствующей версии игры. Ведь PlayStation 5 и Xbox Series похвастают более технологичной картинкой.</div>  
 <div class="bottom\_button">  
 <a href="news.html" class="button\_link">Читать далее</a>  
 </div>  
 </div>  
 </div>  
  
 <div class="in\_box">  
 <div class="box\_top">  
 <div class="top\_img">  
 <a href="" class="img\_link">  
 <img src="https://pbs.twimg.com/media/Elm-s8QX0AAnr4Z.jpg:large" alt="">  
 </a>  
 </div>  
 <div class="top\_title">  
 <a href="" class="title\_link">ВЫШЛА БЕТА-ВЕРСИЯ BLACK MESA DEFINITIVE EDITION, УЛУЧШАЮЩАЯ ГЕЙМПЛЕЙ И ГРАФИКУ</a>  
 </div>  
 </div>  
 <div class="box\_bottom">  
 <div class="bottom\_description">Black Mesa вышла из раннего доступа в марте, однако это не значит, что работа над игрой закончена. Студия Crowbar Collective выпустила открытую бета-версию Definitive Edition — обновления 1.5, которое улучшает геймплей и графику, а также вносит множество точечных исправлений.</div>  
 <div class="bottom\_button">  
 <a href="" class="button\_link">Читать далее</a>  
 </div>  
 </div>  
 </div>  
  
 <div class="in\_box">  
 <div class="box\_top">  
 <div class="top\_img">  
 <a href="" class="img\_link">  
 <img src="https://i.playground.ru/p/X8wVXRQSPwPm1CUTEEL6zA.jpeg" alt="">  
 </a>  
 </div>  
 <div class="top\_title">  
 <a href="" class="title\_link">ЭКШЕН PRAEY FOR THE GODS, ПОХОЖИЙ НА SHADOW OF THE COLOSSUS, ВЫЙДЕТ НА XB1, PS4 И PS5 В НАЧАЛЕ 2021-ГО</a>  
 </div>  
 </div>  
 <div class="box\_bottom">  
 <div class="bottom\_description">Приключенческий боевик Praey for the Gods томится в раннем доступе на PC с начала 2019-го. Коллектив No Matter Studios, состоящий всего из трёх сотрудников, медленно, но верно ведёт свой проект к финишной черте — в первом квартале 2021-го игра дорастёт до версии 1.0 и выйдет на обоих поколениях PlayStation, а также на Xbox One.</div>  
 <div class="bottom\_button">  
 <a href="" class="button\_link">Читать далее</a>  
 </div>  
 </div>  
 </div>  
  
 <div class="in\_box">  
 <div class="box\_top">  
 <div class="top\_img">  
 <a href="" class="img\_link">  
 <img src="https://i.playground.ru/p/DypJ8ZXnSnEAc3uJWmZeeQ.jpeg" alt="">  
 </a>  
 </div>  
 <div class="top\_title">  
 <a href="" class="title\_link">ТОДД ГОВАРД: ТРУДНО ПРЕДСТАВИТЬ, ЧТО MICROSOFT ПОМЕШАЕТ THE ELDER SCROLLS VI ВЫЙТИ НА PLAYSTATION</a>  
 </div>  
 </div>  
 <div class="box\_bottom">  
 <div class="bottom\_description">Компания ZeniMax (и, соответственно, Bethesda) становится частью семейства Microsoft. В воздухе висят животрепещущие вопросы: например, значит ли это, что все последующие игры Bethesda, включая Starfield и The Elder Scrolls VI, окажутся эксклюзивом платформ Microsoft? Тодд Говард (Todd Howard) коснулся этой темы в интервью GamesIndustry.biz.</div>  
 <div class="bottom\_button">  
 <a href="" class="button\_link">Читать далее</a>  
 </div>  
 </div>  
 </div>  
 </div>  
  
 </div>  
  
 <div class="footer">  
 <div class="footer\_in">  
 <div class="in\_button"></div>  
 <div class="in\_social">  
 <div class="social\_button">  
 <a href="">ТАКЖЕ МЫ</a>  
 </div>  
 <div class="social\_in">  
 <div class="in\_box">  
  
 <div class="box\_instagram">  
 <a href="https://www.instagram.com/blog\_games/" class="link\_soc"></a>  
 </div>  
  
 <div class="box\_twitter">  
 <a href="https://twitter.com/games\_news\_" class="link\_soc"></a>  
 </div>  
  
 <div class="box\_vk">  
 <a href="https://vk.com/gamelnews" class="link\_soc"></a>  
 </div>  
  
 <div class="box\_facebook">  
 <a href="https://ru-ru.facebook.com/FacebookGaming" class="link\_soc"></a>  
 </div>  
  
 <div class="box\_telega">  
 <a href="https://tlgrm.ru/channels/gaming" class="link\_soc"></a>  
 </div>  
  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>  
</body>

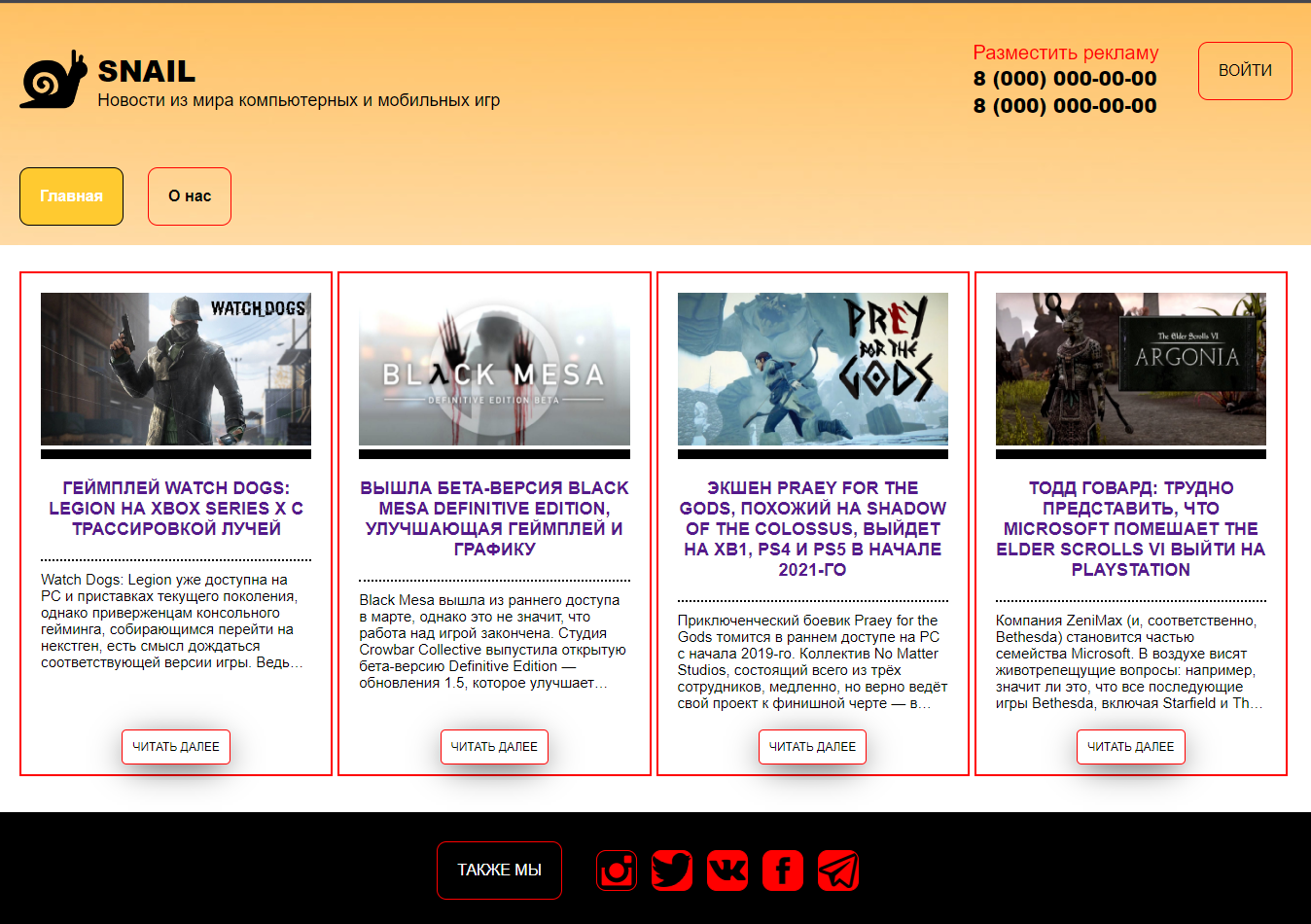
Также использовали настройки стилей и расположения элементов описанный в css-файл.

.site .header {  
 display: flex;  
 flex-direction: column;  
 background: url("img/cropped-back4.jpg") 50% 50% no-repeat;  
 /\*background-size: cover;\*/  
 padding: 0 20px;  
}  
.site .header .header\_top {  
 display: flex;  
 justify-content: space-between;  
 padding: 40px 0 20px;  
 align-items: center;  
}  
.site .header .header\_top .top\_left {  
 display: flex;  
 flex-direction: row;  
 justify-content: space-between;  
 align-items: center;  
}  
.site .header .header\_top .top\_left .left\_logo {  
 width: 70px;  
 padding-right: 10px;  
}  
.site .header .header\_top .top\_left .left\_logo img {  
 width: 100%;  
}  
.site .header .header\_top .top\_left .left\_title {  
 display: flex;  
 flex-direction: column;  
}

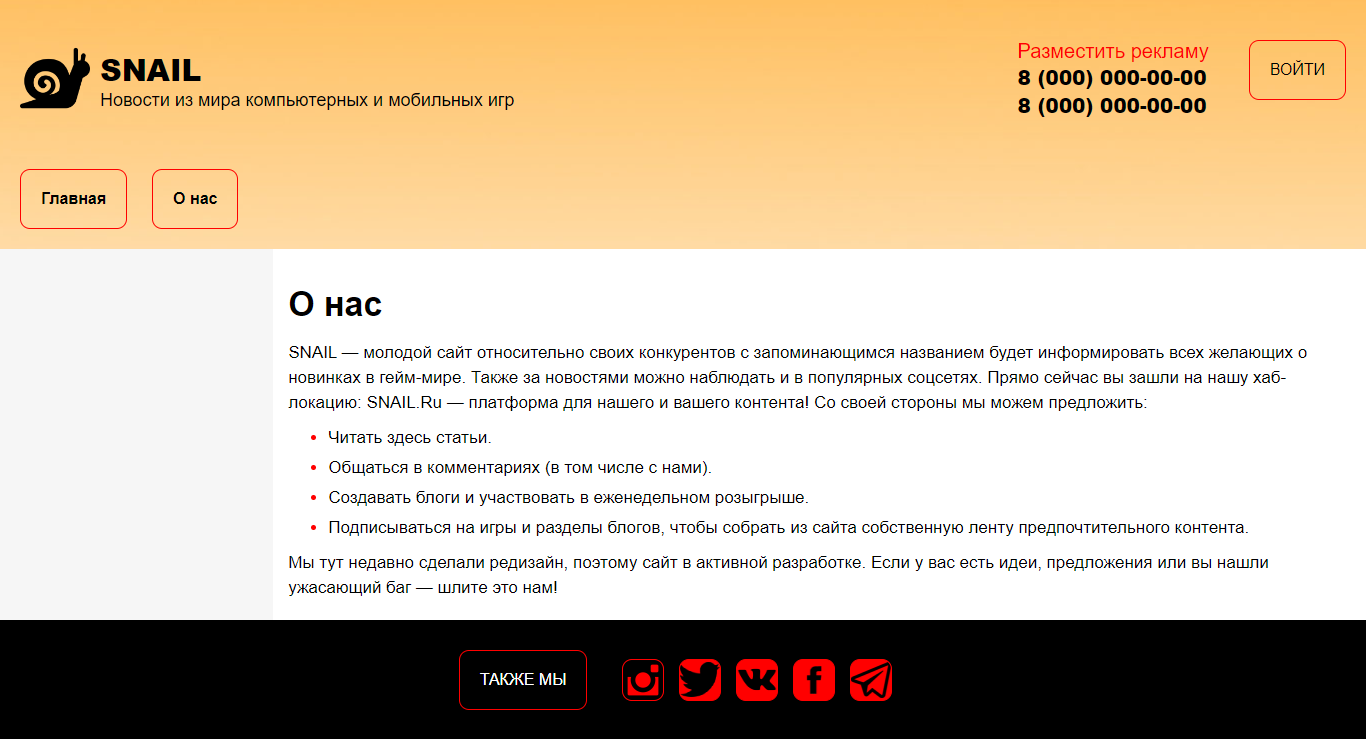
…

### **Полученный результат**

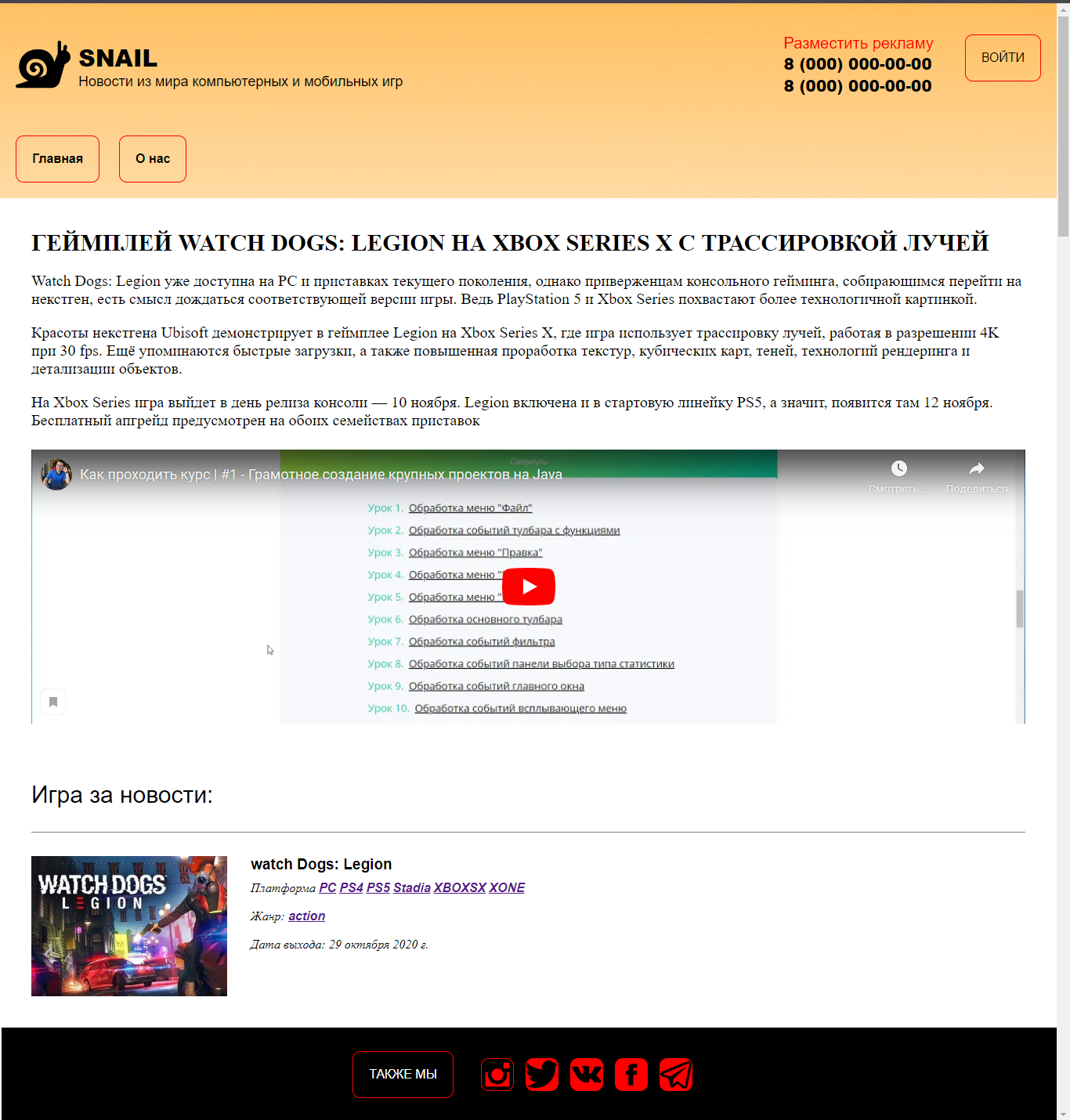
Главная страница



Страница информации о сайте



Страница детального просмотра новости



### Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/webProg.git> -b master

# **Клиентское программирование. React JS**

**Цель**: познакомиться с основами разработки на языке JavaScript. Познакомится с основами разработки frontend приложения, в частности с использованием клиентских фреймворков (React). Научиться переносить вёрстку в отдельные компоненты. Реализовывать ToDoIst сайт, используя технологию React JS

## **Выполнение**

Стек технологий:  
HTML, CSS, JS, REACT, NPM

**React** — это декларативная, эффективная и гибкая библиотека JavaScript для создания пользовательских интерфейсов (UI). Она позволяет вам создавать сложные UI из небольших и изолированных частей кода, называемых «компонентами».

Рассматривая макет из прошлого этапа, то можно

спокойно выделить новостные карточки в один компонент, верхнее меню в другой, нижнее меню аналогично, так как все эти объекты одинаковые.

Подключение необходимых библиотек

{  
 "name": "lab",  
 "version": "0.1.0",  
 "private": true,  
 "dependencies": {  
 "@testing-library/jest-dom": "^5.11.6",  
 "@testing-library/react": "^11.2.2",  
 "@testing-library/user-event": "^12.2.2",  
 "bootstrap": "^4.5.3",  
 "react": "^17.0.1",  
 "react-bootstrap": "^1.4.0",  
 "react-dom": "^17.0.1",  
 "react-router-dom": "^5.2.0",  
 "react-scripts": "4.0.1",  
 "web-vitals": "^0.2.4"  
 },  
 "scripts": {  
 "start": "react-scripts start",  
 "build": "react-scripts build",  
 "test": "react-scripts test",  
 "eject": "react-scripts eject"  
 },  
 "eslintConfig": {  
 "extends": [  
 "react-app",  
 "react-app/jest"  
 ]  
 },  
 "browserslist": {  
 "production": [  
 ">0.2%",  
 "not dead",  
 "not op\_mini all"  
 ],  
 "development": [  
 "last 1 chrome version",  
 "last 1 firefox version",  
 "last 1 safari version"  
 ]  
 }  
}

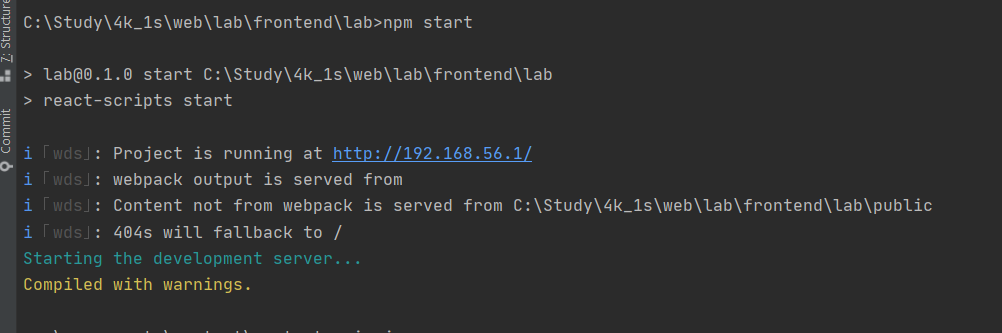
### **Главный компонент**

import ***React*** from "react";  
import { ***BrowserRouter*** as Router, Route, Switch } from "react-router-dom";  
import Content from "../pages/content";  
import Login from "../pages/login";  
import Register from "../pages/register";  
import About from "../pages/about";  
import News from "../pages/news";  
import "./App.css";  
  
function App() {  
 return (  
 <div className="App">  
 <Router>  
 <Switch>  
 <div className="site">  
 <div className="bg">  
 <Route path="/news" component={News} />  
 <Route path="/" exact component={Content} />  
 </div>  
 <Route path="/about" component={About} />  
 <Route path="/login" component={Login} />  
 <Route path="/register" component={Register} />  
 </div>  
 </Switch>  
 </Router>  
 </div>  
 );  
}  
  
export default App;

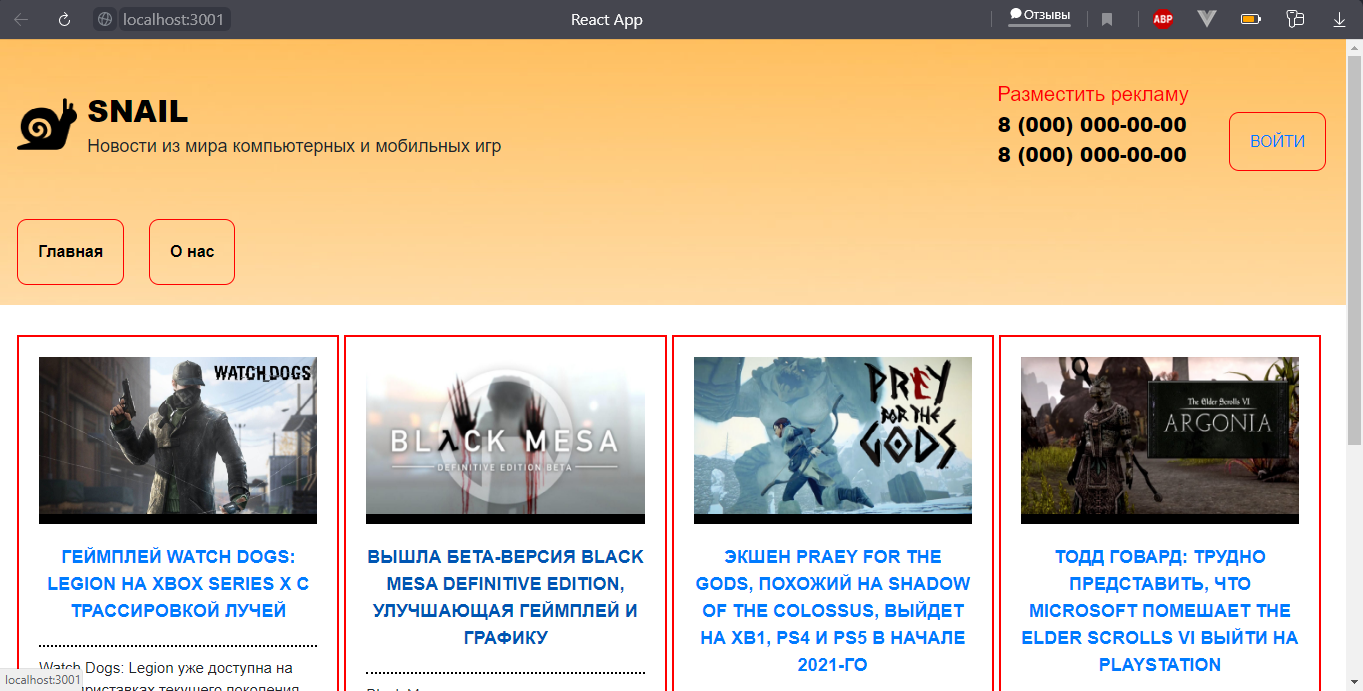
### **Пример кода компонента Header сайта**

import ***React*** from "react";  
import { ***Link*** } from "react-router-dom";  
  
const Header\_menu = () => {  
 return (  
 <div className="header\_menu">  
 <ul className="menu\_in">  
 <li className="in\_after">  
 <a href="/" className="after\_in">  
 Главная  
 </a>  
 <Link to="/about" className="after\_in">  
 О нас  
 </Link>  
 </li>  
 </ul>  
 </div>  
 );  
};  
  
export default Header\_menu;

### **Запуск приложения**



В результате, при переносе верстки на React.js получилось следующее



### Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/webProg.git> -b laba2

# **Серверное программирование. Spring Boot**

**Цель**: познакомиться с основами backend разработки web-приложений. Научится писать программы на языке Java. Познакомиться с основами работы docker. Познакомиться с фреймворком Spring Boot и научиться разворачивать проект, производить его настройку. Научится работать с API в приложении Postman. Реализовывать простой Rest сервис, с двумя endpoints.

## **Выполнение**

На данном этапе мы будем реализовывать простой Rest сервис, с двумя endpoints.

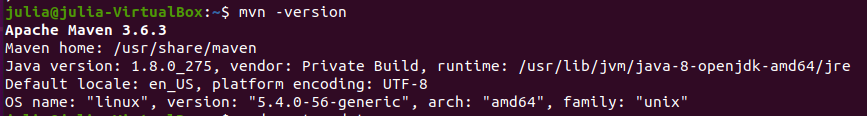
Язык программирования: Java 8

Фреймворк: Spring, Spring Boot

Система сборки: Maven

Контейниризатор: Docker

После установки java8 и maven. Проверим работу maven, узнав его версию.



DockerFile – файл, в котом будет находится конфигурация нашего Docker-образа

FROM openjdk:8  
ADD target/back\_lab3\_web-1.0-SNAPSHOT.jar back\_lab3\_web-1.0-SNAPSHOT.jar  
EXPOSE 8080  
ENTRYPOINT ["java", "-jar", "back\_lab3\_web-1.0-SNAPSHOT.jar"]

pom.xml – файл в котором будут находится все необходимые нам зависимости.

Нам понадобиться библиотека для реализации контролеров, поднятия tomcat.

<plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
</plugin>

### **Создаём точку запуска нашего приложения**

package com;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class App {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(App.class, args);  
 }  
  
}

@SpringBootApplication – аннотация фреймворка, которая делает очень кучу всего. Но нам важно, что она возьмёт свою базовую конфигурацию и настроит зависимости, заинжектид, запустит tomcat на 8080 порту.

### **Создаём контроллеры**

package com.web;  
  
import com.model.News;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.annotation.PostConstruct;  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api")  
public class Controller {  
  
 private List<News> inf = new ArrayList<>();  
  
 @PostConstruct  
 private void init() {  
 for (int i = 0; i < 5; i++) {  
 inf.add(News.*builder*()  
 .title("My title" + i)  
 .description("My description" + i)  
 .build());  
 }  
 }  
  
 @GetMapping("/news")  
 public List<News> getLists() {  
 return inf;  
 }  
  
}

@RestController – аннотация, которая говорит, что этот класс является компонентом и так же является REST контроллером

@RequestMapping("/api") – url этого контроллера начнётся с /test

Над методами находится аннотация @GetMapping("/news") – которая говорит, какой будет тип запроса (GET, POST, DEL и.т.д.) и так же часть его url.

В нашем случае методы будут отдавать json в виде строки.

### **Добавление приложения в Docker**

Docker был установлен на Ubuntu 20.04.

Проверим работу докер командой docker -v

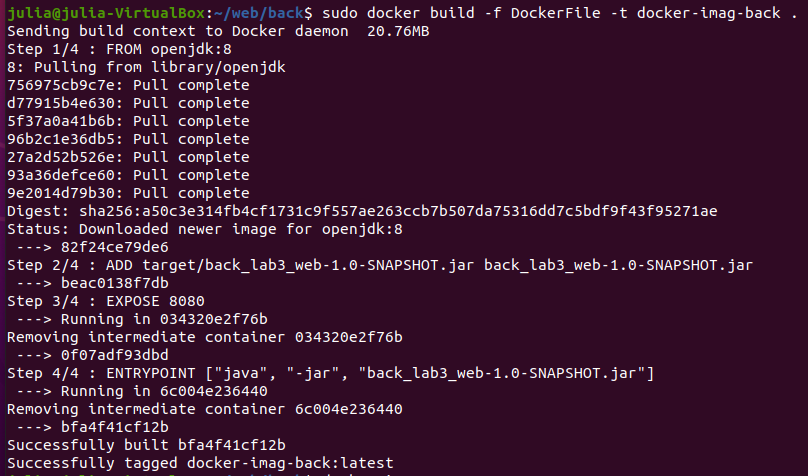


Компилируем приложение и создаём из него jar

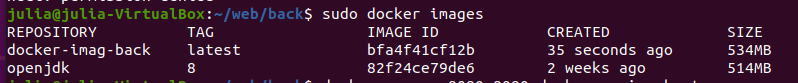


…

Сборка образа с именем docker-imag-back

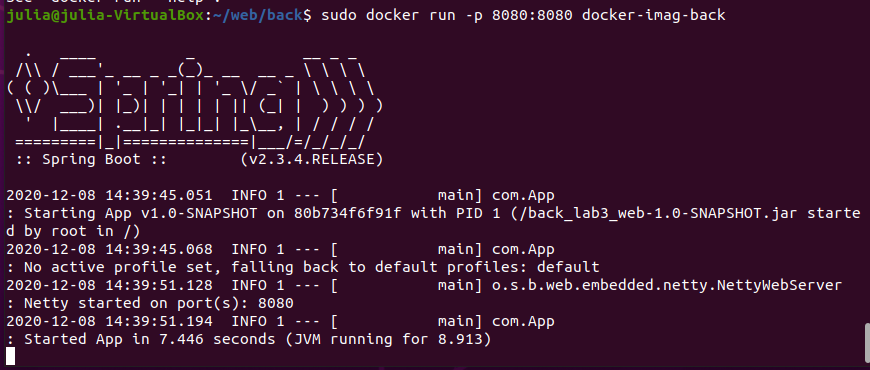


После загрузки всего необходимого докер образа, он должен появиться в списке образов. Проверим это командой docker images



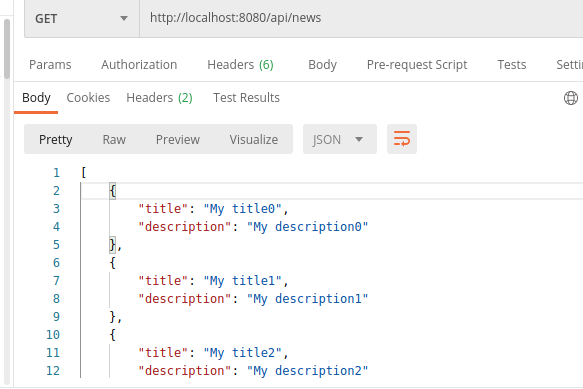
Для запуска образа выполнить команду

sudo docker run -p 8080:8080 docker-imag-back



### **Результат**

http://localhost:8080/api/news





### Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/webBack.git>

# **Разработка и проектирование базы данных**

**Цель**: изучить основы взаимодействия web-приложения с базой данных. Спроектировать базу данных для хранения информации приложения (страницы, пользователи и т.п.). Реализовать бекенд приложение для работы с базой данных с тремя сущностями.

## **Выполнение**

На этом этапе мы будем реализовывать бекенд приложение для работы с базой данных с тремя сущностями

Язык программирования: Java 8

Фреймворк: Spring, Spring Boot, JPA, Hibernate

Система сборки: Maven

База данных: PostgeSQL

После установки java8, maven и PostgeSQL

Создаём новый проект с такой структурой:

pom.xml – файл в котором будут находится все необходимые нам зависимости

App – класс, точка запуска приложения (настройка конфигурации и многое другое)

entity – пакет, в котором будут лежать наши сущности

repository – пакет, в котором будут лежать интефейсы для работы с базой данных.

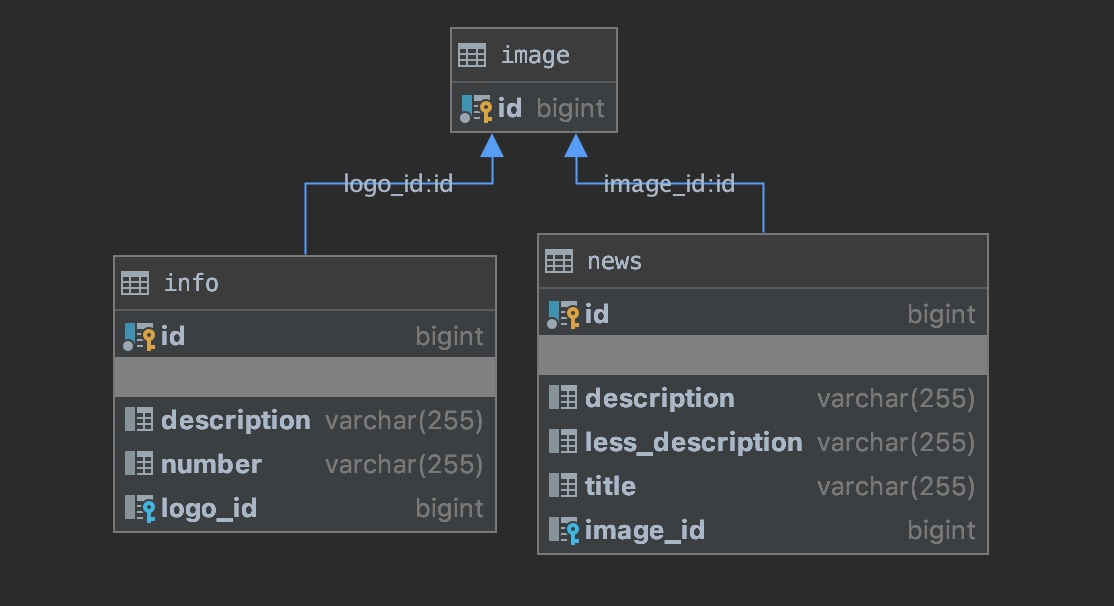
### **Подключение необходимых библиотек**

Нам понадобиться библиотека для работы с PostgreSQL и ORM JPA

<dependencies>  
 . . .  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 </dependencies>

### 

### **Структура базы данных**

****

### **Создание сущностей**

Используя аннотации Hibernate для создания и настройки сущностей задаём имя колонок, связи многие ко многим.

@Data  
@Entity  
@Table(name = "news")  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@EqualsAndHashCode  
public class News {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 Long id;  
  
 @Column(name = "title")  
 String title;  
  
 @Column(name = "less\_description")  
 String lessDescription;  
  
 @Column(name = "description")  
 String description;  
  
 @OneToOne  
 Image image;  
}

@Data  
@Entity  
@Table(name = "info")  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@EqualsAndHashCode  
public class Info {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 Long id;  
  
 @Column(name = "number")  
 String number;  
  
 @OneToOne  
 Image logo;  
  
 @Column(name = "description")  
 String description;  
}

@Data  
@Entity  
@Table(name = "image")  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@EqualsAndHashCode  
public class Image {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(name = "id")  
 Long id;  
}

### **Создание JPA репозиториев**

Мы будем использовать JPA интерфейы, которые под капотом используют ORM Hibernate.   
В этих интерфейсах уже реализованы базовые CRUD операции, а нам больше и не нужно. Так же при запуске приложение hibernate самостоятельно по методанным создаст таблицы в базе данных.  
  
Чтобы создать репозиторий всего лишь нужно создать интерфейс и наследоваться от одного из JPA интерфейсов.

@Repository  
public interface NewsRepository extends JpaRepository<News, Long> {  
}

Так же мы поступаем с остальными сущностями.

### **Работа с PostgeSQL**

Чтобы работать с БД необходимо в конфигурации прописать настройки, куда подключаться.

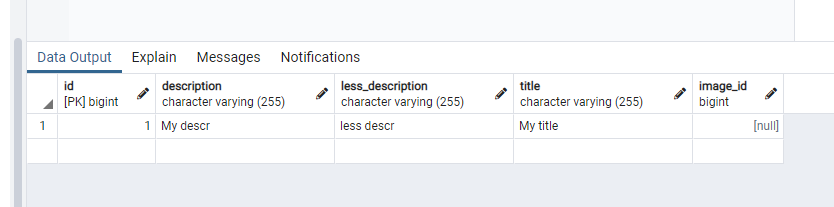
server:  
 port: 8080  
  
spring:  
 datasource:  
 url: jdbc:postgresql://localhost:5432/postgres  
 username: postgres  
 password: 2514  
 jpa:  
 generate-ddl: true  
 show-sql: false

### **Демонстрация работы**

Чтобы продемонстрировать работу приложения был написан тест, который сохраняет данные в базу.

class NewsRepositoryTest extends BaseTest {  
  
 @Autowired  
 private NewsRepository newsRepository;  
  
 @Test  
 void testAddAndReadNewsFromDB() {  
 newsRepository.deleteAll();  
  
 News news = News.*builder*()  
 .description("My descr")  
 .lessDescription("less descr")  
 .title("My title")  
 .build();  
  
 newsRepository.save(news);  
  
 List<News> res = newsRepository.findAll();  
 *assertEquals*("My descr", res.get(0).getDescription());  
 }  
  
}

### **Результат**

****

Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/web_4laba_DB.git>

# **REST API**

**Цель**: изучить основы разработки API для web-приложений. Разработать REST API для своего проекта. Реализовать REST API на стороне бека, основываясь на уже созданной базе данных.

## **Выполнение**

На этом этапе мы будем реализовывать REST API на стороне бека, основываясь на базе данных из прошлого

Язык программирования: Java 8

Фреймворк: Spring, Spring Boot, JPA, Hibernate

Система сборки: Maven

База данных: PostgeSQL

В прошлой лабораторной работе у нас бы были готовы репозитории для работы с нашими сущностями. Теперь нужно написать бизнес-логику и контроллеры для работы с нашим беком.

### **Разработка DTO**

Приходящий у нам JSON нужно мапить в какие-то сущности, для большей простоты работы с данными и конвертации их потом в сущности в базе данных.

Они выглядят например так:

@Data  
@Builder  
public class GameDto {  
  
 Long id;  
  
 String nameGame;  
  
 String platform;  
  
 String genre;  
  
 ImageDto cover;  
  
 String dateReleased;  
}

### **Разработка контроллеров**

Разберём разработку контроллера на примере

@RestController  
@RequestMapping(ApiUtil.*INFO*)  
@RequiredArgsConstructor  
public class InfoController {  
  
 private final InfoService infoService;  
  
 @GetMapping("/getInfo")  
 public InfoDto getInfo() {  
 return infoService.getInfo();  
 }  
}

@RestController – аннотация, которая говорит, что наш контроллер будет получать и передавать данные в формате REST API.

@RequestMapping(ApiUtil.INFO) – так как у нас будет много контроллеров, мы создаём иерархию endpoint-ов.

После мы описываем метод.

@GetMapping("/getInfo ") – аннотация, которая указывает тип запроса и по какому url нужно обратиться к этому методу.

### **Разработка мапперов**

С фронтом мы общаемся через DTO, а с базой данных мы общаемся через сущности. Чтобы преобразовывать одно в другое, мы будем применять мапера, а конкретно библиотеку mupstruct. Мы задаём интерфейс для мапера и объясняем, как поступать в некоторых ситуациях, реализацию библиотека сгенерирует сама.

@Mapper(uses = ImageMapper.class)  
public interface GameMapper {  
  
 GameDto toDto(Game game);  
  
 Game toEntity(GameDto dto);  
}

### **Разработка сервисов**

Слой сервисов – это слой, который реализует бизнес-логику нашего приложения.

Пример:

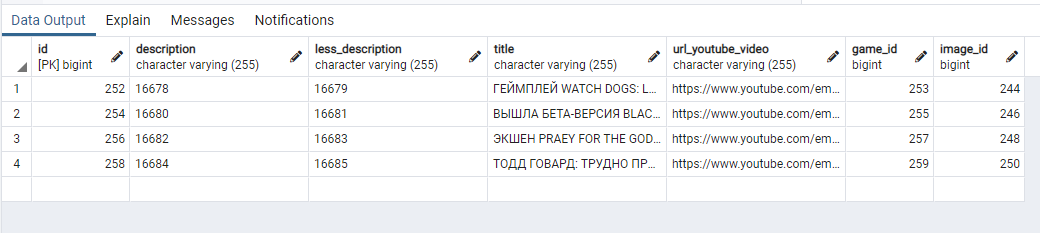
@Service  
@RequiredArgsConstructor  
public class NewsServiceImpl implements NewsService {  
  
 private final NewsRepository newsRepository;  
  
 private final NewsMapper newsMapper;  
  
 @Override  
 public List<NewsDto> getAllNews() {  
 return newsRepository.findAll()  
 .stream().map(newsMapper::toDto)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public NewsDto getNewsById(Long id) {  
 News res = newsRepository.getOne(id);  
  
 return newsMapper.toDto(res);  
 }  
}

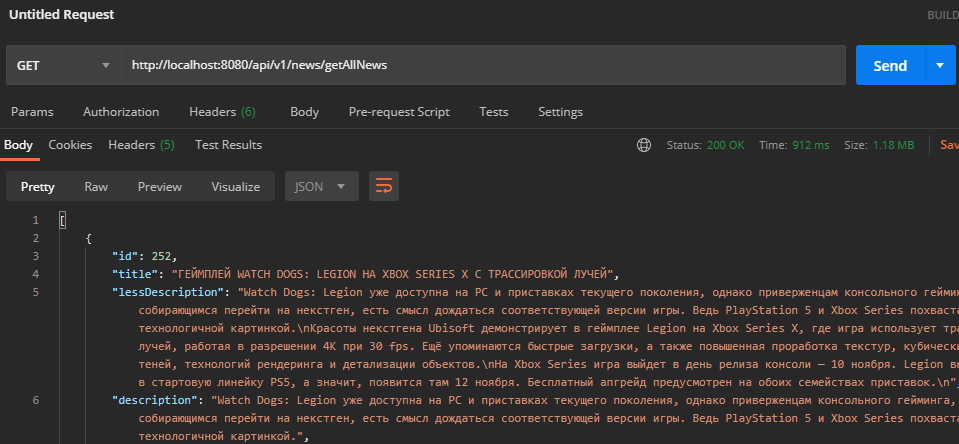
Сервис решает, что во что мапить, что сохранять в базу и так далее.

### 

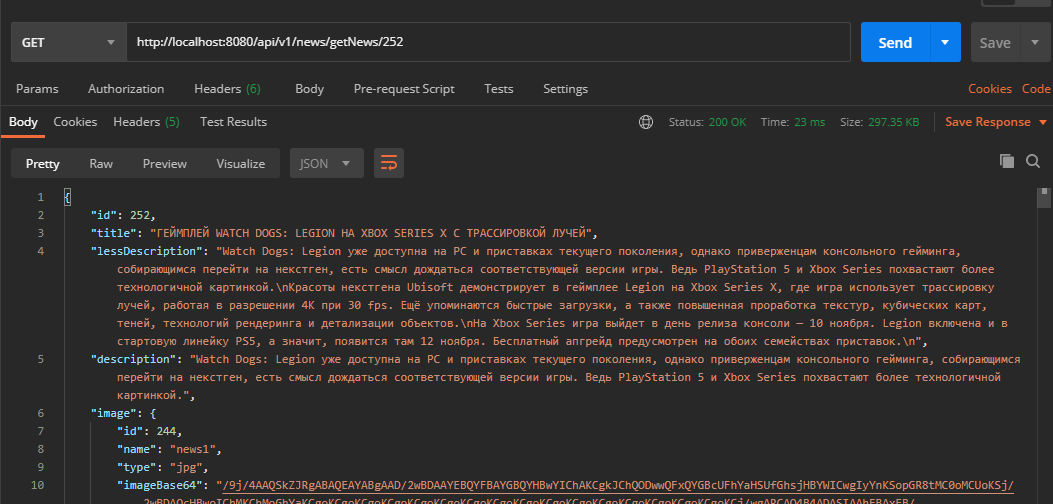
### **Пример работы с REST API**

Продемонстрируем запрос на получение всех всех новостей.

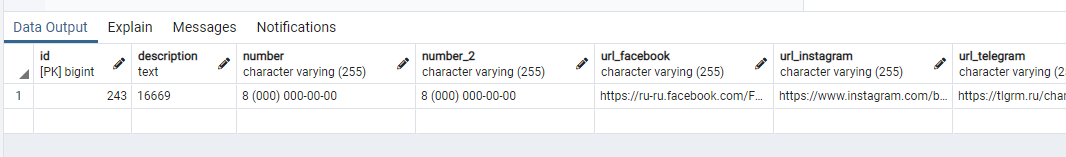


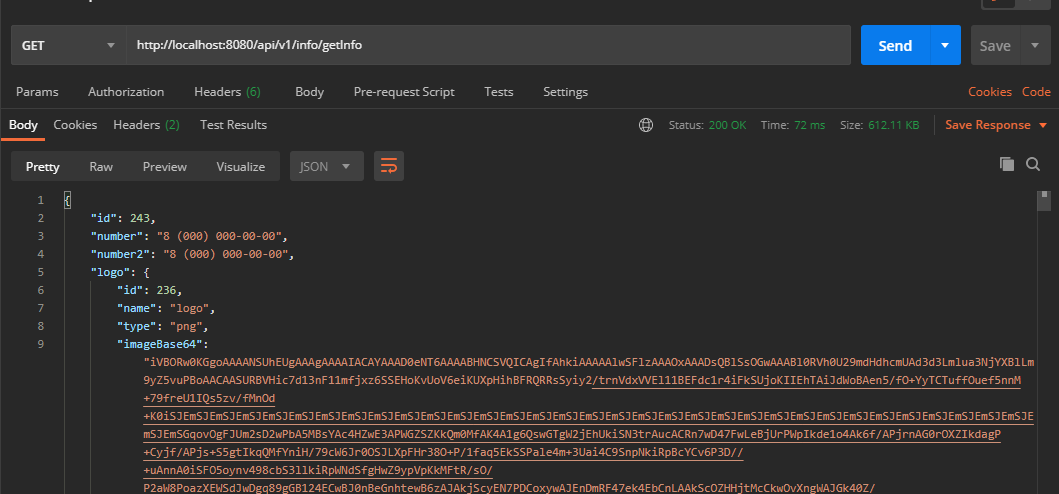


Продемонстрируем запрос на получение новостей по id.

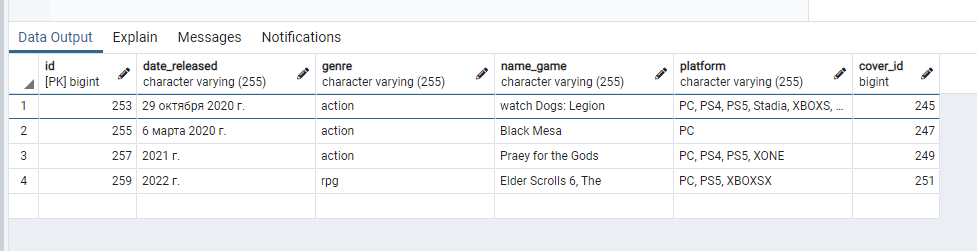


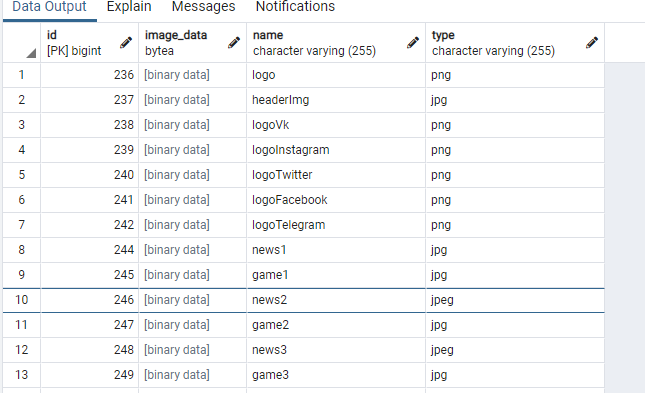
Продемонстрируем запрос на получение информации о сайте.





Состояние других таблиц





Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/web_laba5.git> -b master

# **Работа с HTTP запросами**

**Цель**: изучить принципы работы отправки Ajax запросов. Изучить работу отправки синхронных и асинхронных запросов в React. Получить навыки работы с промисами. Необходимо связать фронт и бек из предыдущих лабораторных работ.

## **Выполнение**

В приложении предполагается следующее: имеется заполненная база данных новостей. Пользователь переходить на страницы новостей, в бд хранится статический набор новостей.

## **Back**

Для реализации http запросов были разработаны контроллеры

InfoController

@RestController  
@RequestMapping(ApiUtil.*INFO*)  
@RequiredArgsConstructor  
@CrossOrigin  
public class InfoController {  
  
 private final InfoService infoService;  
  
 @GetMapping("/getInfo")  
 public List<InfoDto> getInfo() {  
 //return infoService.getInfo();  
 return Collections.*singletonList*(infoService.getInfo());  
 }  
}

NewsController

@RestController  
@RequestMapping(ApiUtil.*NEWS*)  
@CrossOrigin  
@RequiredArgsConstructor  
@Transactional  
public class NewsController {  
  
 private final NewsService newsService;  
  
 @GetMapping("/getAllNews")  
 public List<NewsDto> getAllNews() {  
 return newsService.getAllNews();  
 }  
  
 @GetMapping("/getNews/{id}")  
 public NewsDto getNewsById(@PathVariable("id") Long id) {  
 return newsService.getNewsById(id);  
 }  
}

ForDBController

@RestController  
@RequestMapping(ApiUtil.*DB*)  
@RequiredArgsConstructor  
@CrossOrigin  
public class ForDBController {  
  
 private final ImageRepository imageRepository;  
  
 private final NewsRepository newsRepository;  
  
 private final InfoRepository infoRepository;  
  
  
 @GetMapping("/clean")  
 @Transactional  
 public void clearDb() {  
 newsRepository.deleteAll();  
 infoRepository.deleteAll();  
 imageRepository.deleteAll();  
 }  
  
 @GetMapping("/fill")  
 @Transactional  
 public void fillDB() {  
 addAllInfoToDB();  
 addAllNewsInDB();  
 }  
  
 void addAllNewsInDB() {  
 newsRepository.deleteAll();  
 List<News> allNews = new ArrayList<>();  
  
 News news1 = News.*builder*()  
 .title("ГЕЙМПЛЕЙ WATCH DOGS: LEGION НА XBOX SERIES X С ТРАССИРОВКОЙ ЛУЧЕЙ")  
 .lessDescription("Watch Dogs: Legion уже доступна на PC и приставках текущего поколения, однако приверженцам консольного гейминга, собирающимся перейти на некстген, есть смысл дождаться соответствующей версии игры. Ведь PlayStation 5 и Xbox Series похвастают более технологичной картинкой.\n" +  
 "Красоты некстгена Ubisoft демонстрирует в геймплее Legion на Xbox Series X, где игра использует трассировку лучей, работая в разрешении 4K при 30 fps. Ещё упоминаются быстрые загрузки, а также повышенная проработка текстур, кубических карт, теней, технологий рендеринга и детализации объектов.\n" +  
 "На Xbox Series игра выйдет в день релиза консоли — 10 ноября. Legion включена и в стартовую линейку PS5, а значит, появится там 12 ноября. Бесплатный апгрейд предусмотрен на обоих семействах приставок.\n"  
 )  
 .description("Watch Dogs: Legion уже доступна на PC и приставках текущего поколения, однако приверженцам консольного гейминга, собирающимся перейти на некстген, есть смысл дождаться соответствующей версии игры. Ведь PlayStation 5 и Xbox Series похвастают более технологичной картинкой.")  
 .image(addImage("src/main/resources/news/news1.jpg", "news1", "jpg"))  
 .urlYouTubeVideo("https://www.youtube.com/embed/imhDZGe\_bwI")  
 .game(  
 Game.*builder*()  
 .nameGame("watch Dogs: Legion")  
 .platform("PC, PS4, PS5, Stadia, XBOXS, XXONE")  
 .genre("action")  
 .cover(addImage("src/main/resources/games/game1.jpg", "game1", "jpg"))  
 .dateReleased("29 октября 2020 г.")  
 .build())  
 .build();  
  
 News news2 = News.*builder*()  
 .title("ВЫШЛА БЕТА-ВЕРСИЯ BLACK MESA DEFINITIVE EDITION, УЛУЧШАЮЩАЯ ГЕЙМПЛЕЙ И ГРАФИКУ")  
 .lessDescription("Black Mesa вышла из раннего доступа в марте, однако это не значит, что работа над игрой закончена. Студия Crowbar Collective выпустила открытую бета-версию Definitive Edition — обновления 1.5, которое улучшает геймплей и графику, а также вносит множество точечных исправлений.\n" +  
 "Нововведения касаются в том числе следующих глав:\n" +  
 "Power Up. Главу изменили так, чтобы сделать геймплей более «причинно-следственным», а общую цель — более ясной. Теперь вам нужно включить свет, убить Гаргантюа и активировать вагонетку. Во время прохождения можно оценить новую систему динамического освещения.\n" +  
 "On A Rail. Разработчики внесли крупномасштабные изменения в ракетную шахту, дабы лучше показать задачу и её последующее решение, благодаря чему глава стала более цельной.\n" +  
 "Кроме того, авторы продолжают подтягивать внешний вид локаций до уровня мира Зен.\n"  
 )  
 .description("Black Mesa вышла из раннего доступа в марте, однако это не значит, что работа над игрой закончена. Студия Crowbar Collective выпустила открытую бета-версию Definitive Edition — обновления 1.5, которое улучшает геймплей и графику, а также вносит множество точечных исправлений.\n")  
 .image(addImage("src/main/resources/news/news2.jpeg", "news2", "jpeg"))  
 .urlYouTubeVideo("https://www.youtube.com/embed/VfUU8Y0CG7E")  
 .game(  
 Game.*builder*()  
 .nameGame("Black Mesa")  
 .platform("PC")  
 .genre("action")  
 .cover(addImage("src/main/resources/games/game2.jpg", "game2", "jpg"))  
 .dateReleased("6 марта 2020 г.")  
 .build())  
 .build();  
  
 News news3 = News.*builder*()  
 .title("ЭКШЕН PRAEY FOR THE GODS, ПОХОЖИЙ НА SHADOW OF THE COLOSSUS, ВЫЙДЕТ НА XB1, PS4 И PS5 В НАЧАЛЕ 2021-ГО")  
 .lessDescription("Приключенческий боевик Praey for the Gods томится в раннем доступе на PC с начала 2019-го. Коллектив No Matter Studios, состоящий всего из трёх сотрудников, медленно, но верно ведёт свой проект к финишной черте — в первом квартале 2021-го игра дорастёт до версии 1.0 и выйдет на обоих поколениях PlayStation, а также на Xbox One.\n" +  
 "Разумеется, на PS5 экшен будет выгодно отличаться от PS4-версии. Особенности у издания для некстгена следующие:\n" +  
 "60 fps (против 30 на PS4).\n" +  
 "Максимальные графические настройки — полные тени, текстуры высокого разрешения и так далее.\n" +  
 "Значительно ускоренные загрузки.\n" +  
 "Поддержка адаптивных триггеров и тактильной отдачи DualSense.\n" +  
 "Бесплатный апгрейд с PS4.\n"  
 )  
 .description("Приключенческий боевик Praey for the Gods томится в раннем доступе на PC с начала 2019-го. Коллектив No Matter Studios, состоящий всего из трёх сотрудников, медленно, но верно ведёт свой проект к финишной черте — в первом квартале 2021-го игра дорастёт до версии 1.0 и выйдет на обоих поколениях PlayStation, а также на Xbox One.")  
 .image(addImage("src/main/resources/news/news3.jpeg", "news3", "jpeg"))  
 .urlYouTubeVideo("https://www.youtube.com/embed/xQMn4JEZmiE")  
 .game(  
 Game.*builder*()  
 .nameGame("Praey for the Gods")  
 .platform("PC, PS4, PS5, XONE")  
 .genre("action")  
 .cover(addImage("src/main/resources/games/game3.jpg", "game3", "jpg"))  
 .dateReleased("2021 г.")  
 .build())  
 .build();  
  
 News news4 = News.*builder*()  
 .title("ТОДД ГОВАРД: ТРУДНО ПРЕДСТАВИТЬ, ЧТО MICROSOFT ПОМЕШАЕТ THE ELDER SCROLLS VI ВЫЙТИ НА PLAYSTATION")  
 .lessDescription("Компания ZeniMax (и, соответственно, Bethesda) становится частью семейства Microsoft. В воздухе висят животрепещущие вопросы: например, значит ли это, что все последующие игры Bethesda, включая Starfield и The Elder Scrolls VI, окажутся эксклюзивом платформ Microsoft? Тодд Говард (Todd Howard) коснулся этой темы в интервью GamesIndustry.biz.\n" +  
 "В целом Microsoft лояльно относится к другим платформам: скажем, Minecraft выходит вообще везде, где только можно, а сделки о временной эксклюзивности Deathloop и GhostWire: Tokyo для PlayStation 5 остаются в силе. «Трудно представить, что Microsoft, несмотря на всё своё влияние над Bethesda, ограничит The Elder Scrolls VI своими платформами», — пишет интервьюер GI.biz.\n" +  
 "«Соглашусь, представить такое трудно», — подтверждает Тодд Говард. Впрочем, не рассматривайте это как гарантию того, что TESVI появится на PlayStation: формально слияние Microsoft и Bethesda пока не завершено, а некоторые детали сделки ещё обсуждаются. Говард признаётся, что и сам пока не знает, на каких условиях выйдут Starfield и TESVI: в данный момент лишь подтверждено, что их добавят в Xbox Game Pass прямо во время релиза.\n" +  
 "В то же время Говард вспоминает, что в истории The Elder Scrolls полно примеров тесного сотрудничества с Microsoft: Morrowind выходила на первой Xbox, Oblivion долгое время оставалась консольным эксклюзивом Xbox 360 (то есть появлялась на PC, но не на PlayStation 3), а DLC к Skyrim стартовали в первую очередь у «мелкомягких».\n"  
 )  
 .description("Компания ZeniMax (и, соответственно, Bethesda) становится частью семейства Microsoft. В воздухе висят животрепещущие вопросы: например, значит ли это, что все последующие игры Bethesda, включая Starfield и The Elder Scrolls VI, окажутся эксклюзивом платформ Microsoft? Тодд Говард (Todd Howard) коснулся этой темы в интервью GamesIndustry.biz.")  
 .image(addImage("src/main/resources/news/news4.jpeg", "news4", "jpeg"))  
 .urlYouTubeVideo("https://www.youtube.com/embed/OkFdqqyI8y4")  
 .game(  
 Game.*builder*()  
 .nameGame("Elder Scrolls 6, The")  
 .platform("PC, PS5, XBOXSX")  
 .genre("rpg")  
 .cover(addImage("src/main/resources/games/game4.jpg", "game4", "jpg"))  
 .dateReleased("2022 г.")  
 .build())  
 .build();  
  
 allNews.add(news1);  
 allNews.add(news2);  
 allNews.add(news3);  
 allNews.add(news4);  
 newsRepository.saveAll(allNews);  
 }  
  
 void addAllInfoToDB() {  
 infoRepository.deleteAll();  
  
 Info info = Info.*builder*()  
 .number("8 (000) 000-00-00")  
 .number2("8 (000) 000-00-00")  
 .logo(addImage("src/main/resources/info/snail.png","logo" ,"png"))  
 .headerImg(addImage("src/main/resources/info/back.jpg","headerImg" ,"jpg"))  
 .logoVk(addImage("src/main/resources/info/vk.png", "logoVk", "png"))  
 .logoInstagram(addImage("src/main/resources/info/inst.png", "logoInstagram", "png"))  
 .logoTwitter(addImage("src/main/resources/info/twit.png", "logoTwitter", "png"))  
 .logoFacebook(addImage("src/main/resources/info/faceb1.png", "logoFacebook", "png"))  
 .logoTelegram(addImage("src/main/resources/info/teleg.png", "logoTelegram", "png"))  
 .urlVk("https://vk.com/gamelnews")  
 .urlInstagram("https://www.instagram.com/blog\_games/")  
 .urlTwitter("https://twitter.com/games\_news\_")  
 .urlFacebook("https://ru-ru.facebook.com/FacebookGaming")  
 .urlTelegram("https://tlgrm.ru/channels/gaming")  
 .description("SNAIL — молодой сайт относительно своих конкурентов с запоминающимся названием будет информировать всех желающих о новинках в гейм-мире. Также за новостями можно наблюдать и в популярных соцсетях. Прямо сейчас вы зашли на нашу хаб-локацию: SNAIL.Ru — платформа для нашего и вашего контента! Со своей стороны мы можем предложить:\n" +  
 "Читать здесь статьи.\n" +  
 "Общаться в комментариях (в том числе с нами).\n" +  
 "Создавать блоги и участвовать в еженедельном розыгрыше.\n" +  
 "Подписываться на игры и разделы блогов, чтобы собрать из сайта собственную ленту предпочтительного контента.\n" +  
 "Мы тут недавно сделали редизайн, поэтому сайт в активной разработке. Если у вас есть идеи, предложения или вы нашли ужасающий баг — шлите это нам!")  
 .build();  
  
 infoRepository.save(info);  
 }  
  
  
 private Image addImage(String path, String nameImg, String typeImg) {  
 File file = new File(path);  
 byte[] bFile = new byte[(int) file.length()];  
  
 try {  
 FileInputStream fileInputStream = new FileInputStream(file);  
 fileInputStream.read(bFile);  
 fileInputStream.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 Image image = Image.*builder*()  
 .name(nameImg) // имя картинки  
 .type(typeImg) // тип  
 .imageData(bFile)  
 .build();  
  
 return imageRepository.save(image);  
 }  
}

**Front**

Для этого мы будем использовать инструмент Fetch API.

Для получения результата запроса был реализован сервис

Service.js

export default class RestApi {  
 constructor() {  
 this.\_apiBase = "http://localhost:8081/api/v1/";  
 }  
  
 getAllNews = async (url) => {  
 const result = await fetch(`${this.\_apiBase}${url}`);  
 if (!result.ok) {  
 throw new ***Error***(`Could not fetch ${url}, status : ${result.status}`);  
 }  
 return await result.json();  
 };  
  
 getNews = async (id) => {  
 const result = this.getAllNews(`news/getAllNews/${id}`);  
 return result;  
 };  
  
 getInfo = async (url) => {  
 const result = await fetch(`${this.\_apiBase}${url}`);  
 if (!result.ok) {  
 throw new ***Error***(`Could not fetch ${url}, status : ${result.status}`);  
 }  
 return await result.json();  
 };  
}

Метод fetch() — современный и очень мощный, поэтому начнём с него. Он не поддерживается старыми (можно использовать полифил), но поддерживается всеми современными браузерами.

Браузер сразу же начинает запрос и возвращает промис, который внешний код использует для получения результата.

Процесс получения ответа обычно происходит в два этапа.

Во-первых, promise выполняется с объектом встроенного класса Response в качестве результата, как только сервер пришлёт заголовки ответа.

На этом этапе мы можем проверить статус HTTP-запроса и определить, выполнился ли он успешно, а также посмотреть заголовки, но пока без тела ответа.

Промис завершается с ошибкой, если fetch не смог выполнить HTTP-запрос, например при ошибке сети или если нет такого сайта. HTTP-статусы 404 и 500 не являются ошибкой.

Во-вторых, для получения тела ответа нам нужно использовать дополнительный вызов метода.

Для получения ответа в виде json используем await result.json()

пример компонента news

import ***React***, { useState, useEffect } from "react";  
import RestApi from "../service";  
  
const apiReq = new RestApi();  
  
const News\_main = ({ id }) => {  
 const [state, setState] = useState({ newArr: [] });  
  
 useEffect(() => {  
 apiReq.getAllNews("news/getAllNews").then((item) => {  
 return setState(() => {  
 return searchElem(***Number***(id), item);  
 });  
 });  
 }, []);  
  
 const searchElem = (id, item) => {  
 const index = item.findIndex((elem) => elem.id === id);  
 const newArr = item.slice(index, index + 1);  
 return { newArr };  
 };  
  
 ***console***.log(state.newArr);  
 return state.newArr.map((item) => {  
 const { id, title, lessDescription, urlYouTubeVideo, game } = item;  
 return (  
 <div key={id}>  
 <div className="block\_text">  
 <b>{title}</b>  
 <p>{lessDescription}</p>  
 <iframe  
 width="100%"  
 height="350px"  
 src={urlYouTubeVideo}  
 frameborder="0"  
 allow="accelerometer; autoplay; encrypted-media; gyroscope; picture-in-picture"  
 allowfullscreen  
 ></iframe>  
 </div>  
 <div className="news\_block">  
 <div className="wrap\_top">Игра за новости:</div>  
 <div className="wrap\_bottom">  
 <div className="bottom\_left">  
 <img  
 src={`data:image/jpeg;base64,${game.cover.imageBase64}`}  
 alt={game.cover.name}  
 />  
 </div>  
 <div className="bottom\_right">  
 <div className="right\_title">{game.nameGame}</div>  
 <div className="right\_description">  
 <p>Платформа</p>  
 <a href="">{game.platform}</a>;  
 <br />  
 <br />  
 Жанр:  
 <a href="">{game.genre}</a>  
 </div>  
 <div className="right\_data">Дата выхода: {game.dateReleased}</div>  
 </div>  
 </div>  
 </div>  
 </div>  
 );  
 });  
};  
  
export default News\_main;

### **Возможные проблемы**

При тестировании работы я столкнулась с такой нюансом, как работа с CORS.

**Cross-origin resource** — технология современных браузеров, которая позволяет предоставить веб-странице доступ к ресурсам другого домена.

Сама технология достаточно проста. Есть три домена, желающие загрузить ресурсы с сервера Z. Для того чтобы это стало возможным, веб-серверу Z, который отдает контент, достаточно указать в заголовке ответа **Access-Control-Allow-Origin** список доверенных доменов: A, B, C. Тогда для страниц этих доменов не будут действовать ограничения принципа одинакового источника на запрашиваемые страницы:

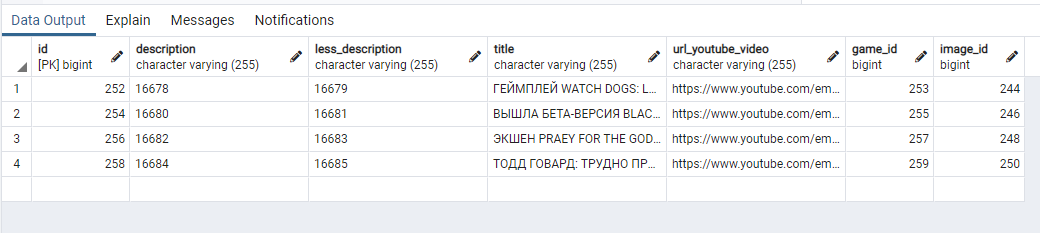
После этого страницы доменов A, B, C смогут загружать контент с сервера Z.

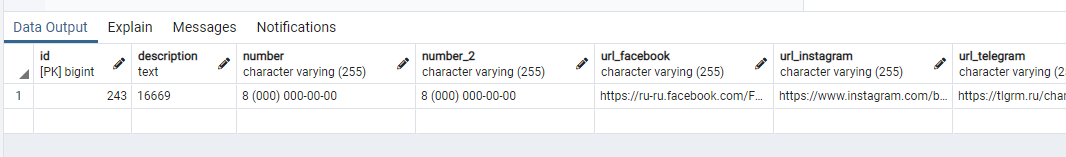
### **Решение проблемы**

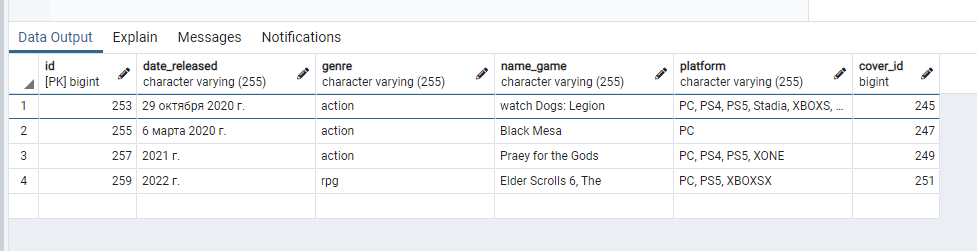
Работу с этим ограничением нужно реализовать со стороны сервера.

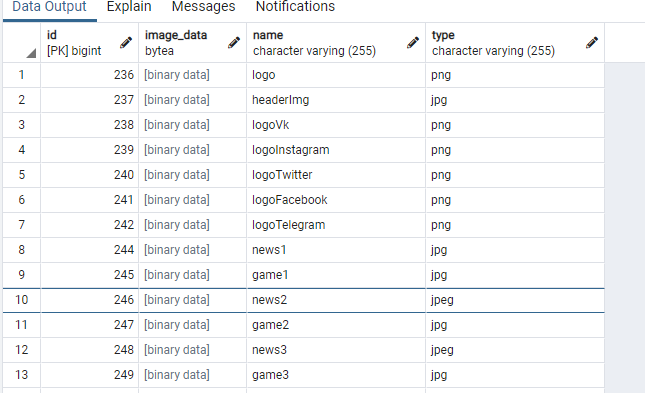
Spring нам предоставляет нам стандартную реализацию настройки этого ограничения. Для этого нам необходимо на наши контроллеры навесить аннотации **@CrossOrigin**, которая по стандартной настройки будет пускать всех. Этот вариант нам вполне подходит.

### **Состояние базы данных**

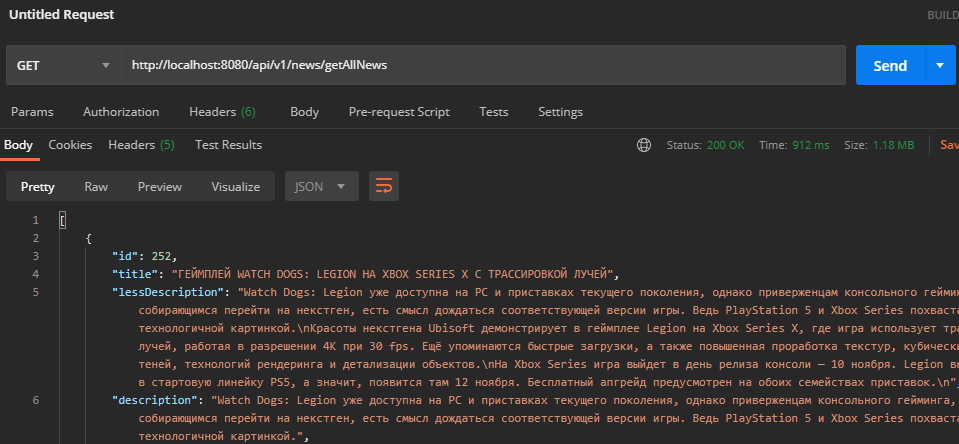


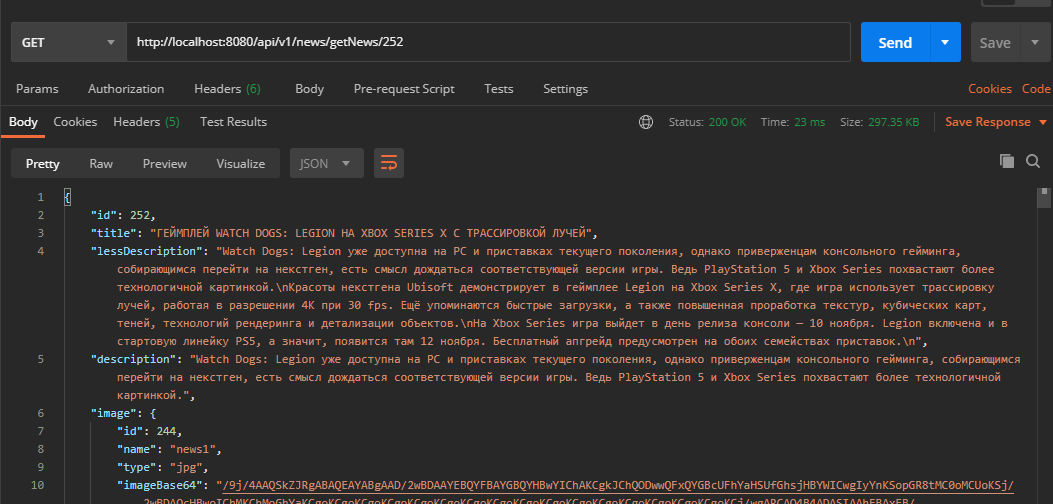


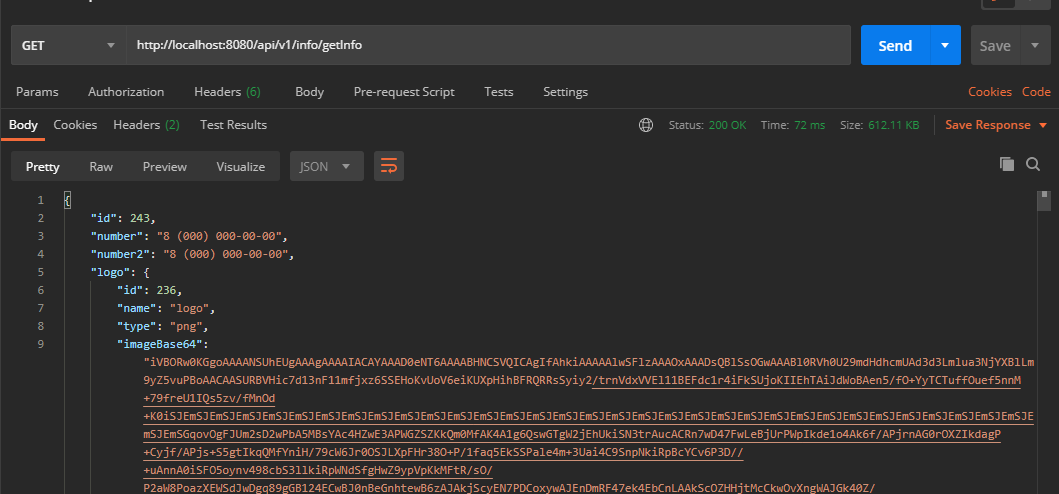




### **Результаты выполнения GET-запросов**

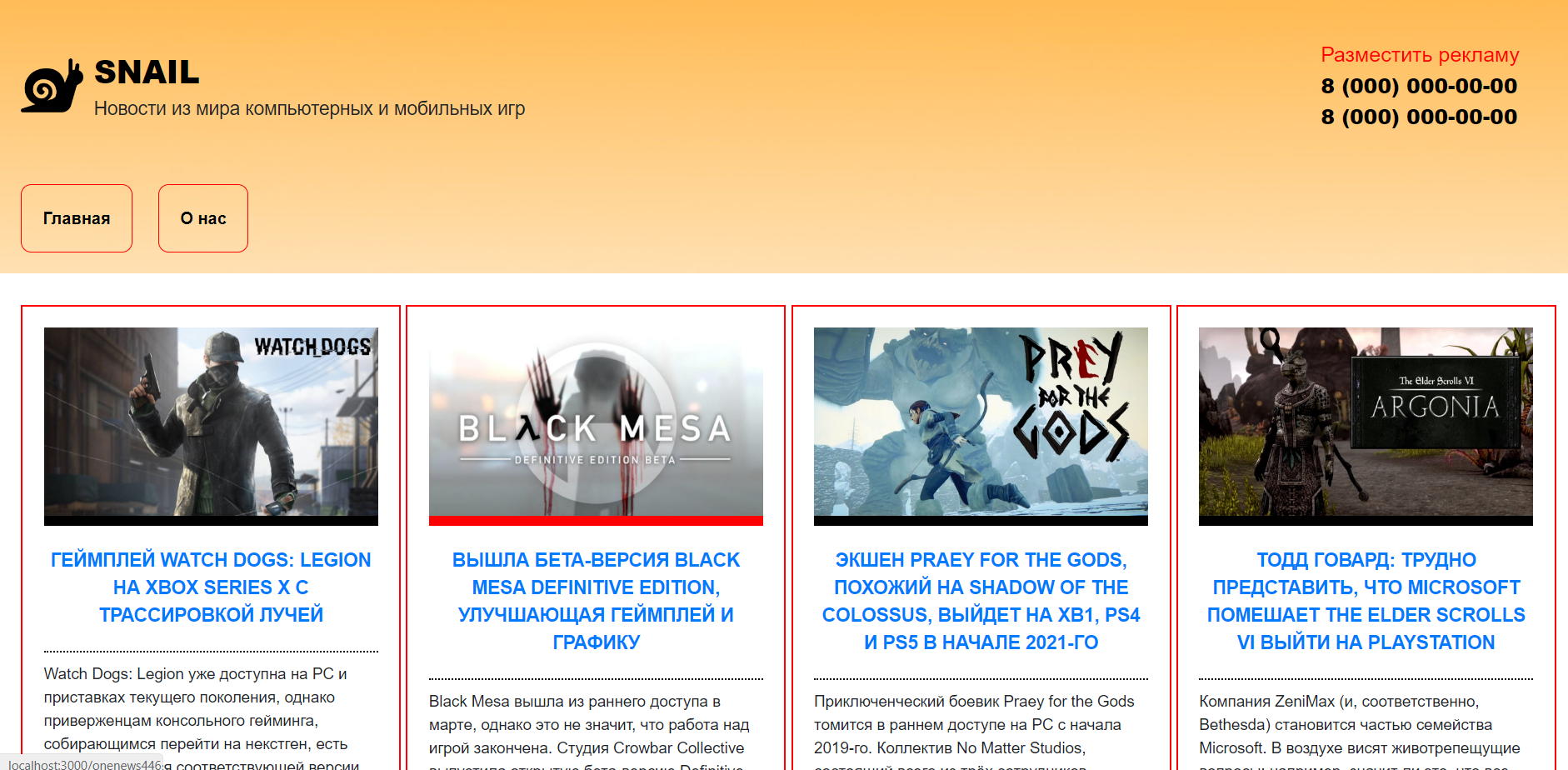




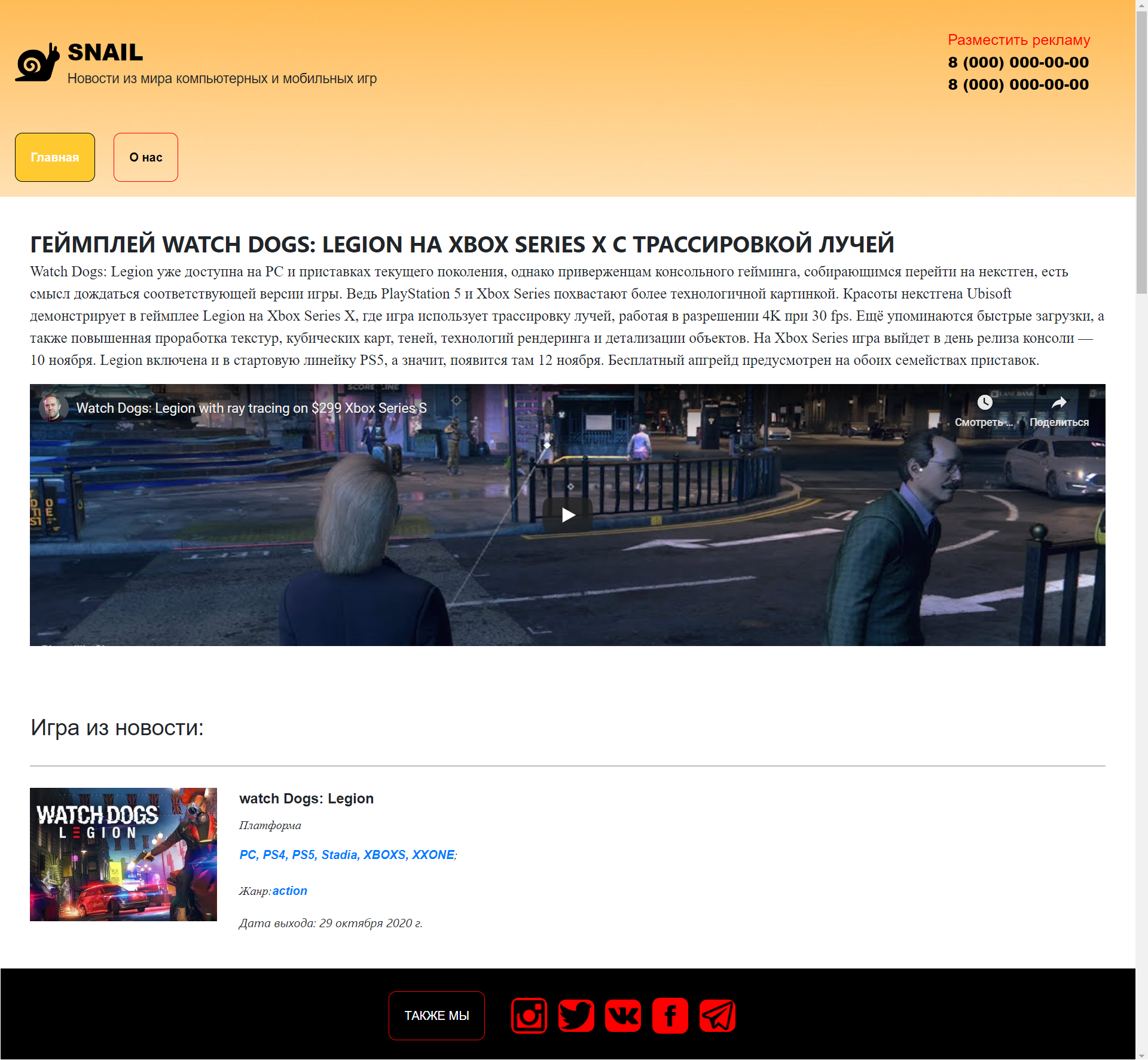


### **Общий вид получившегося сайта**

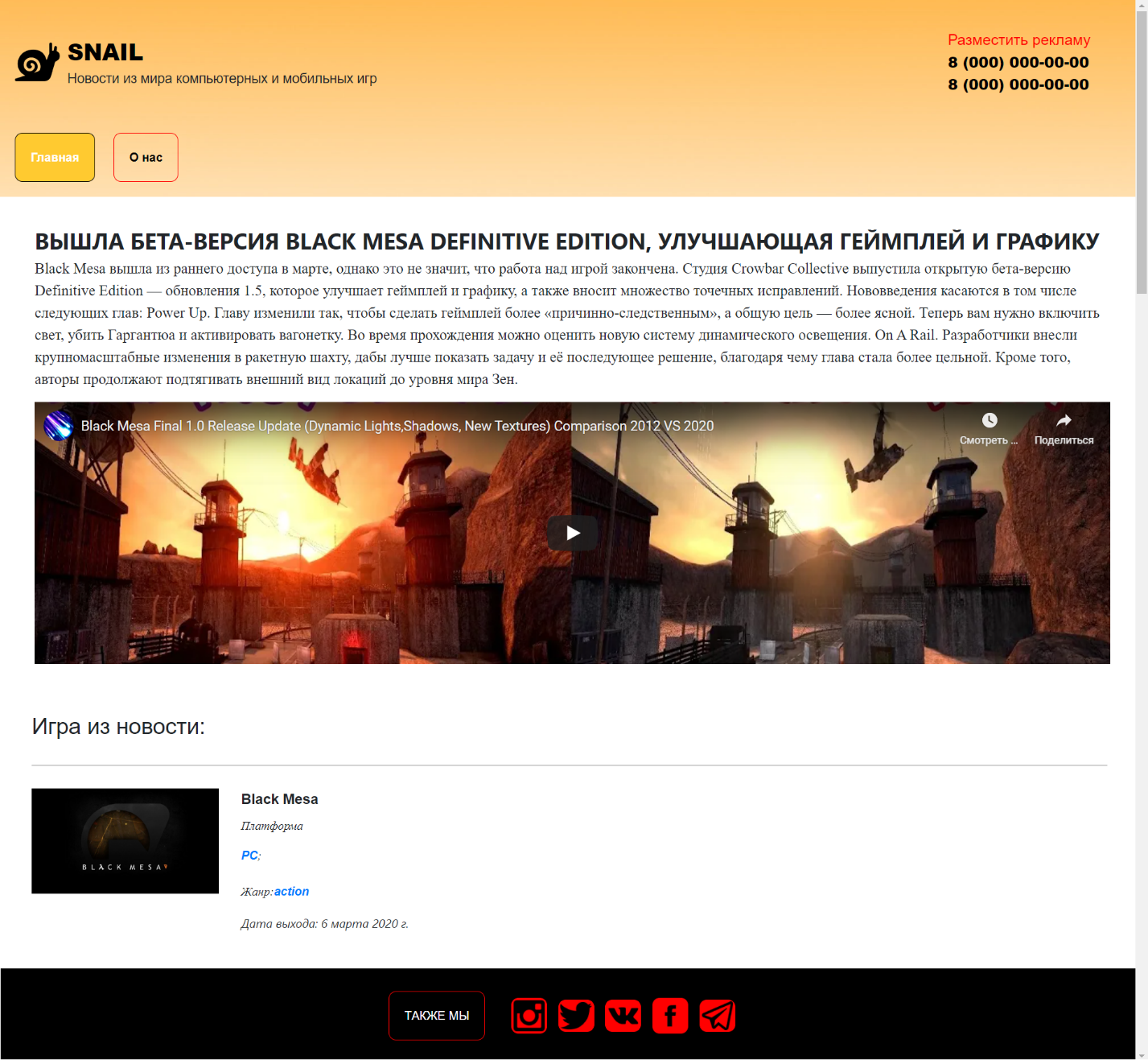
Главная страница



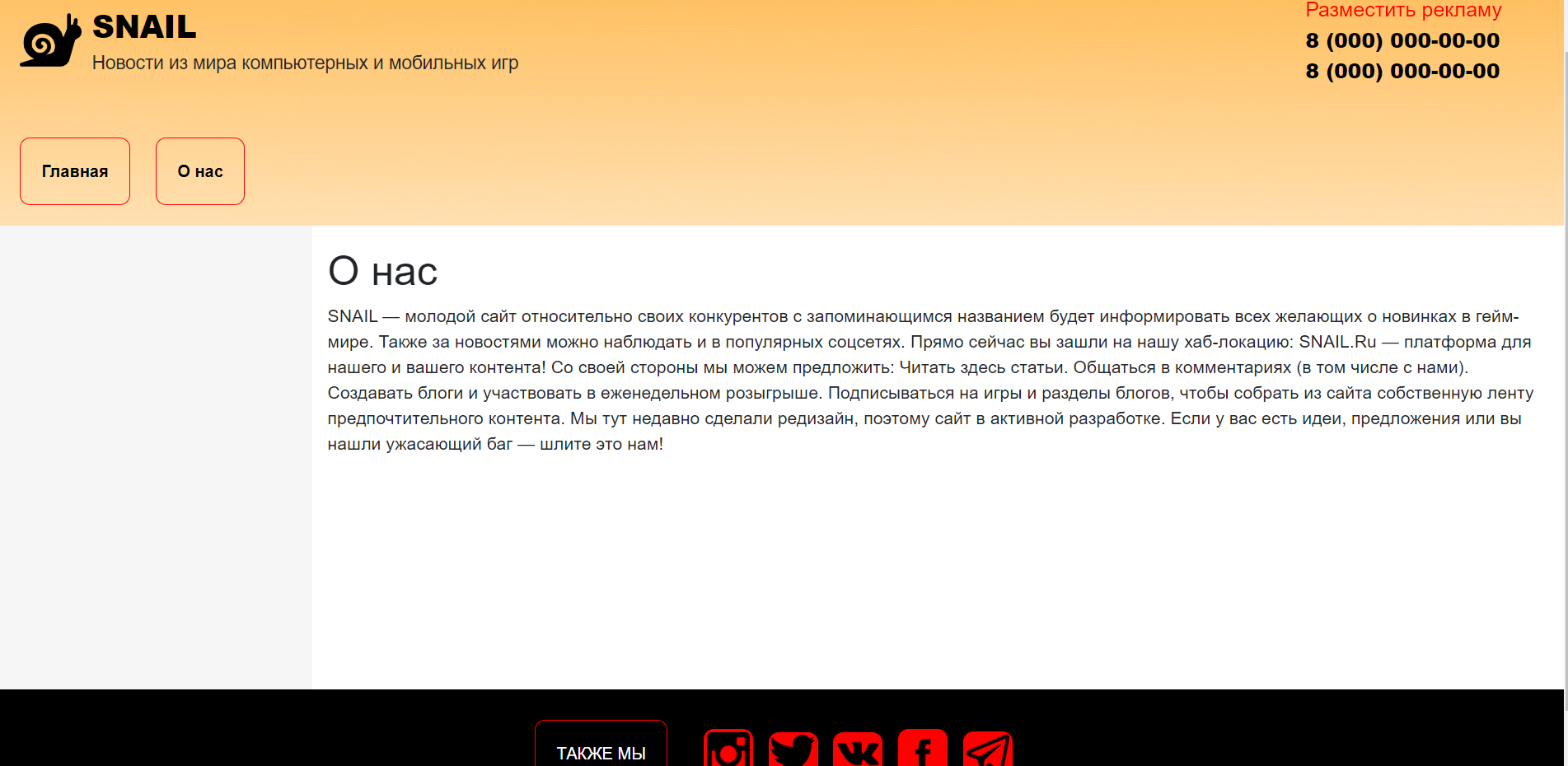
Страница детального просмотра новости 1



Страница детального просмотра новости 2



Страница информации о сайте



### Весь код можно посмотреть по ссылке:

<https://github.com/Julia1999000/webApp.git>

# **Вывод**

В данной работе мы прошли полный путь разработки web-приложения. Научились писать бек, фронт, взаимодействие между ними, всё собирать в образы, автоматически запускать. Было изучено много актуальных и полезных технологий.

Изучили основы разметки страниц с помощью языка HTML. Изучили основы вёрстки страниц, адаптированных под различные устройства. Изучили основы работы задания стилей для разметки страницы с помощью языка SCSS/CSS. Разработали макеты и «сверстали» страницы для web-приложения.

Познакомились с основами разработки на языке JavaScript. Познакомились с основами разработки frontend-приложений, в частности, с использованием клиентских фреймворков (React.js). Научились переносить вёрстку в отдельные компоненты.

Познакомились с основами backend-разработки web-приложений. Познакомились с основами работы в docker. Научились работать с API в приложении Postman.

Изучили основы взаимодействия web-приложения с базой данных, спроектировали базу данных для хранения информации приложения.

изучили основы разработки API для web-приложений и разработали REST API для своего проекта, заполнили с помощью него базу данных.

Изучили принципы работы отправки запросов, изучили принцип работы отправки запросов в react.js.