**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ СОЦИОКУЛЬТУРНЫХ КОММУНИКАЦИЙ**

**Кафедра информационных технологий**

**РАЗРАБОТКА САЙТА КОМАНДЫ ГЕЙМ-РАЗРАБОТЧИКОВ «TRIANGLE»**

**Курсовой проект**

|  |  |
| --- | --- |
|  | Кострицы Анны Александровны  студентки 4 курса,  специальности  «Прикладная информатика»  Научный руководитель:  старший преподаватель  Царик Сергей Всеволодович |

Допущен к защите

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ…………………………………………………..…………………....…](#_heading=h.1fob9te)3

[ГЛАВА 1.](#_heading=h.3znysh7) ОБЗОР ТЕХНОЛОГИЙ И СРЕДСТВ РАЗРАБОТКИ……....…….…...4

[1.1.](#_heading=h.2et92p0) Цели и задачи сайта гейм-разработчиков.........………..…………….....……...[4](#_heading=h.3znysh7)

[1.2.](#_heading=h.2et92p0) Обзор технологий и средств для разработки..…...…………..……….............[4](#_heading=h.2et92p0)

[ГЛАВА](#_heading=h.17dp8vu) 2. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ САЙТА....................7

[2.1](#_heading=h.3rdcrjn) Разработка дизайна сайта «Triangle»...........………………………..……….....7

[2.2](#_heading=h.26in1rg) Разработка клиентской части сайта..……………….....………………….........9

[ГЛАВА](#_heading=h.17dp8vu) 3. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ САЙТА……..………….........12

3.1 Проектирование структуры базы данных…………………...………….........[12](#_heading=h.3rdcrjn)

3[.2](#_heading=h.26in1rg) Разработка серверной части сайта………………....…………………………[1](#_heading=h.26in1rg)5

[ЗАКЛЮЧЕНИЕ……………………………………………………..………….......23](#_heading=h.44sinio)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ……………..………………..24](#_heading=h.2jxsxqh)

[ПРИЛОЖЕНИ](#_heading=h.z337ya)Е[……………..…………](#_heading=h.2jxsxqh)…………………………………………...25

**ВВЕДЕНИЕ**

Сайт для молодого постепенно развивающегося бизнеса является основной платформой для получения информации о том, что из себя представляет команда и чем она занимается. По факту это своеобразная визитная карточка в Интернете, которая быстро поможет получить данные о компании. Разработчикам компьютерных и мобильных игр сайт-визитка позволяет быстро рассказать о разрабатываемых играх, о выпущенных играх, а также о составе команды и о инструментах, которые используются при разработке.

Задачей сайта является для привлечение потенциальных кандидатов, заинтересованных в гейм-разработке, которые в последующем могут стать членами команды, а также партнеров и инвесторов, готовых вложиться материально в развитие команды и бизнеса.

Целью курсового проекта является разработка сайта-визитки для молодой команды разработчиков компьютерных и мобильных игр. Это поможет быстро продемонстрировать преимущества команды, а также познакомить с разрабатываемыми продуктами. Для достижения цели были поставлены следующие задачи:

* провести брифинг с командой с целью сбора информации, необходимой для разработки сайта;
* провести анализ сайтов-аналогов, которые были выбраны командой в качестве образцов для подражания;
* спроектировать прототип и разработать дизайн сайта;
* проанализировать технологии и выбрать инструменты для разработки;
* разработать сайт.

**ГЛАВА 1. АНАЛИТИЧЕСКИЙ ОБЗОР И ПОСТАНОВКА ЗАДАЧИ**

**1.1. Цели и задачи сайта для гейм-разработчиков**

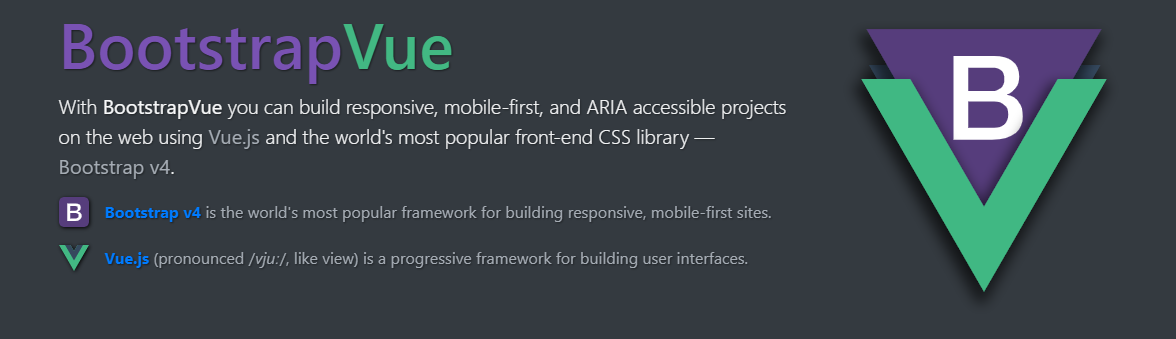
При создании сайта в первую очередь нужно определиться с его целью о создания и с задачами, которые сайт должен выполнять. В будущем цель и задачи создания сайта помогут определить эффективность разработанного проекта. Сайт-визитка для команды разработчиков игр должен рассказывать в первую очередь о творческих проектах команды, которые уже разработаны или же находятся в разработке. Это позволит посетителям быстро оценить работу команды, а также протестировать игры. Кроме этого сайт необходимо дополнить контактами и способом для обратной связи на случай, если кто-то (потенциальный претендент в члены команды или инвестор) захочет сотрудничать с разработчиками.

Было принято решение разработать сайт-визитку в виде посадочной страницы с отображением информации о разработках команды, ведением блога о разработке игр, блоком с описанием членов команды, а также контактов. При разработке сайта поставлены следующие задачи:

* разработать главную страницу в стиле команды;
* добавить блок с разрабатываемыми играми с ссылками на скачивание;
* создать блок со статьями и отдельную страницу подробного прочтения;
* добавить блок с персональными карточками разработчиков;
* разработать блок контактов и форму обратной связи;
* разработать админку для удобства добавления новой информации на сайт.

**1.2. Обзор технологий и средств для разработки**

При верстке страниц сайта использовался язык разметки и таблицы стилей HTML и CSS. Для легкости и оперативности создания шаблонов использовалась библиотека BootstrapVue (рисунок 3.1).



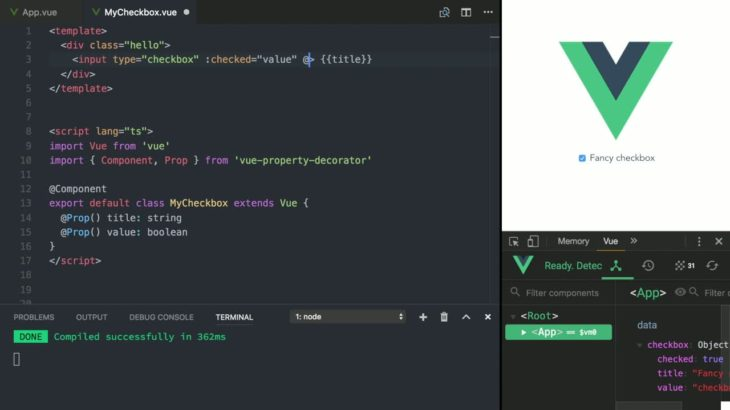
*Рисунок 3.1 – «Библиотека BootstrapVue»*

Библиотека BootstrapVue используется для создания адаптивных сайтов и приложений, разрабатываемых с помощью фреймворка Vue.js [3]. Преимущества данной библиотеки следующие:

* содержит наиболее полную систему компонентов, директив и сетки Bootstrap v4.4 доступных с Vue.js;
* позволяет адаптировать интерфейс под любые устройства;
* имеет открытый исходный код;
* позволяет создавать пользовательские темы и стили с помощью SCSS и настраивать элементы при необходимости;
* легко интегрируется с другими современными фреймворками и библиотеками (например Nuxt.js или Vue-storefront.js) при указании тех компонентов, директив или плагинов, которые необходимы после внедрения.

При разработке клиентской части интерфейса информационного модуля использовался Vue.js – фреймворк, написанный на языке Javascript [4] для создания пользовательских интерфейсов (рисунок 3.2). Ядро данного фреймворка решает задачи уровня представления, и это упрощает интеграцию с другими библиотеками и уже созданными ранее проектами [5]. Vue.js обладает следующими преимуществами в сравнении с другими фреймворками:

* автоматическое отслеживание компонентов во время отрисовки, что делает программный код более производительным;
* разбиение компонента на шаблон на HTML и написанный для него скрипт, что облегчает читаемость программного кода и использование различных возможностей Vue.js;
* обеспечение полного доступа к CSS в том же файле, где находится остальная часть программного кода компонента;
* простота в изучении, открытый исходный программный код, широкая документация с примерами использования.

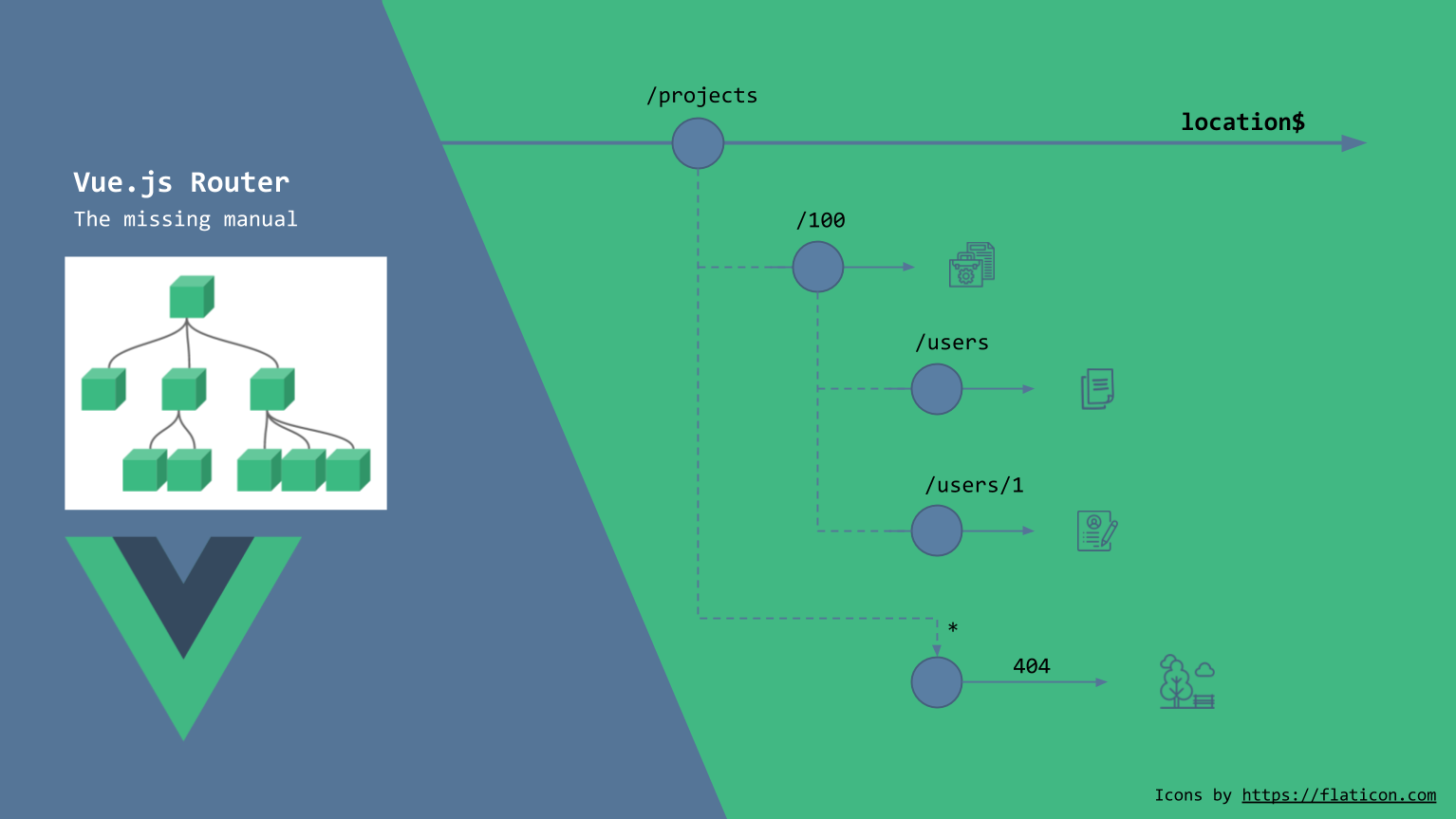


*Рисунок 3.2 – «Фреймворк Vue.js»*

Внедрить фреймворк Vue.js можно с помощью Node.js или скачать напрямую файлы скриптов, положить их в папку с проектом и прописать в файле верстки путь к файлу.

В разработке информационного модуля использовалась библиотека VueRouter.js, написанные на языке Javascript, которая позволяет создавать вложенные маршруты и представления, интегрируется с Vue.js (рисунок 3.3). Преимуществами библиотеки VueRouter.js являются:

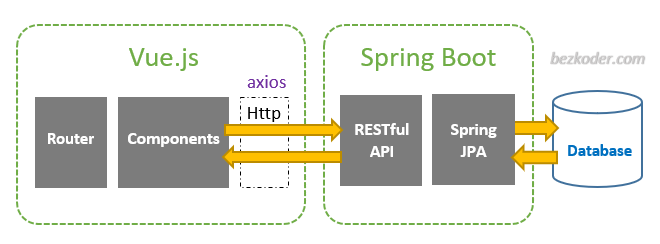
* удобный контроль навигации;
* настраиваемое поведение прокрутки страницы;
* создание вложенных маршрутов и представлений, а также дочерних маршрутов;
* анимация переходов представлений на основе Vue.js [6].



*Рисунок 3.3 – «Схема работы с библиотекой VueRouter.js»*

Создавать одностраничные приложения, используя Vue.js и VueRouter.js удобно, так как при структурировании приложения из компонентов, происходит сопоставление с указанными в главном крипте маршрутами для каждого из компонентов – роутинг.

В курсовом проекте кроме библиотек и фреймворков JavaScript использовался язык программирования Java при написании контроллеров и сущностей при разработке серверной части сайта. Была использована среда Spring Boot (рисунок 3.4) с открытым исходным кодом, используемая для создания микро сервиса [8].



*Рисунок 3.4 – «Работа Spring Boot с Vue.js»*

Микросервис – это архитектура, которая позволяет разработчикам самостоятельно разрабатывать и развертывать сервисы. Spring Boot предлагает следующие преимущества для своих разработчиков:

* Легко понять и разработать весенние приложения
* Увеличивает производительность
* Сокращает время разработки
* Spring Boot разработан для следующих целей:
* Чтобы избежать сложной конфигурации XML в Spring
* Разрабатывать готовые приложения Spring проще
* Сократить время разработки и запустить приложение самостоятельно
* Предложите более простой способ начать работу с приложением [7].

Создавая микросервис на основе Spring необходимо совершить следующие настройки:

* В зависимости от типа создаваемого приложения (Spring MVC, Spring JDBC) импортировать необходимые Spring-модули.
* Импортировать библиотеку web-контейнеров.
* Импортировать необходимые сторонние библиотеки
* Конфигурировать компоненты web-слоя такие, как: диспетчер ресурсов, view resolver.
* Определить класс, который загрузит все необходимые конфигурации.

**ГЛАВА 2. РЕАЛИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОЙ ЧАСТИ САЙТА**

**2.1 Разработка дизайна сайта «Triangle»**

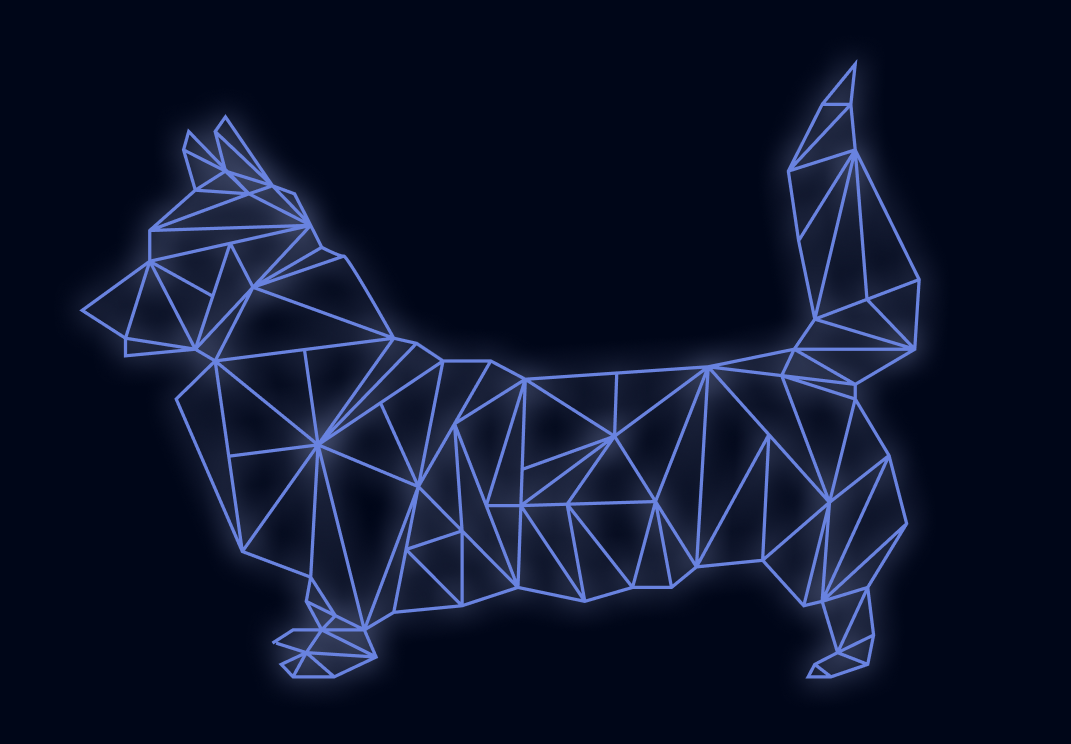
Дизайн сайта проектировался сразу же, без разработки черно-белого прототипа, так как структура была ясна и продумана заранее. Необходимо было определиться с цветовой палитрой и также фирменным стилем команды. За основу была взята палитра в синих оттенках, а также разработан логотип команды с названием «Triangle». Кроме того, у команды есть символ – собака породы корги, поэтому была отрисована минималистичная корги с помощью полигонов. Цветовая палитра, логотип и символ представлены на рисунках 2.3, 2.4 и 2.5.



*Рисунок 2.3 – «Цветовая палитра сайта»*



*Рисунок 2.4 – «Логотип команды с названием»*

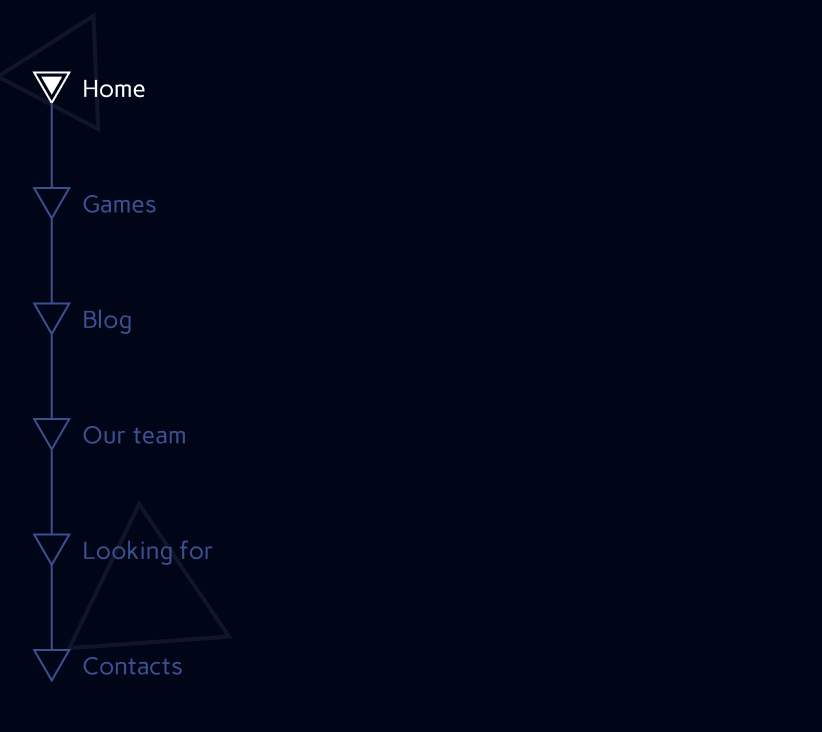


*Рисунок 2.5 – «Символ команды гейм-разработчиков»*

После этого были собраны макеты в Figma – инструменте для создания прототипов и дизайна сайтов. Макет главной страницы представлен в приложении А. На главной странице расположены:

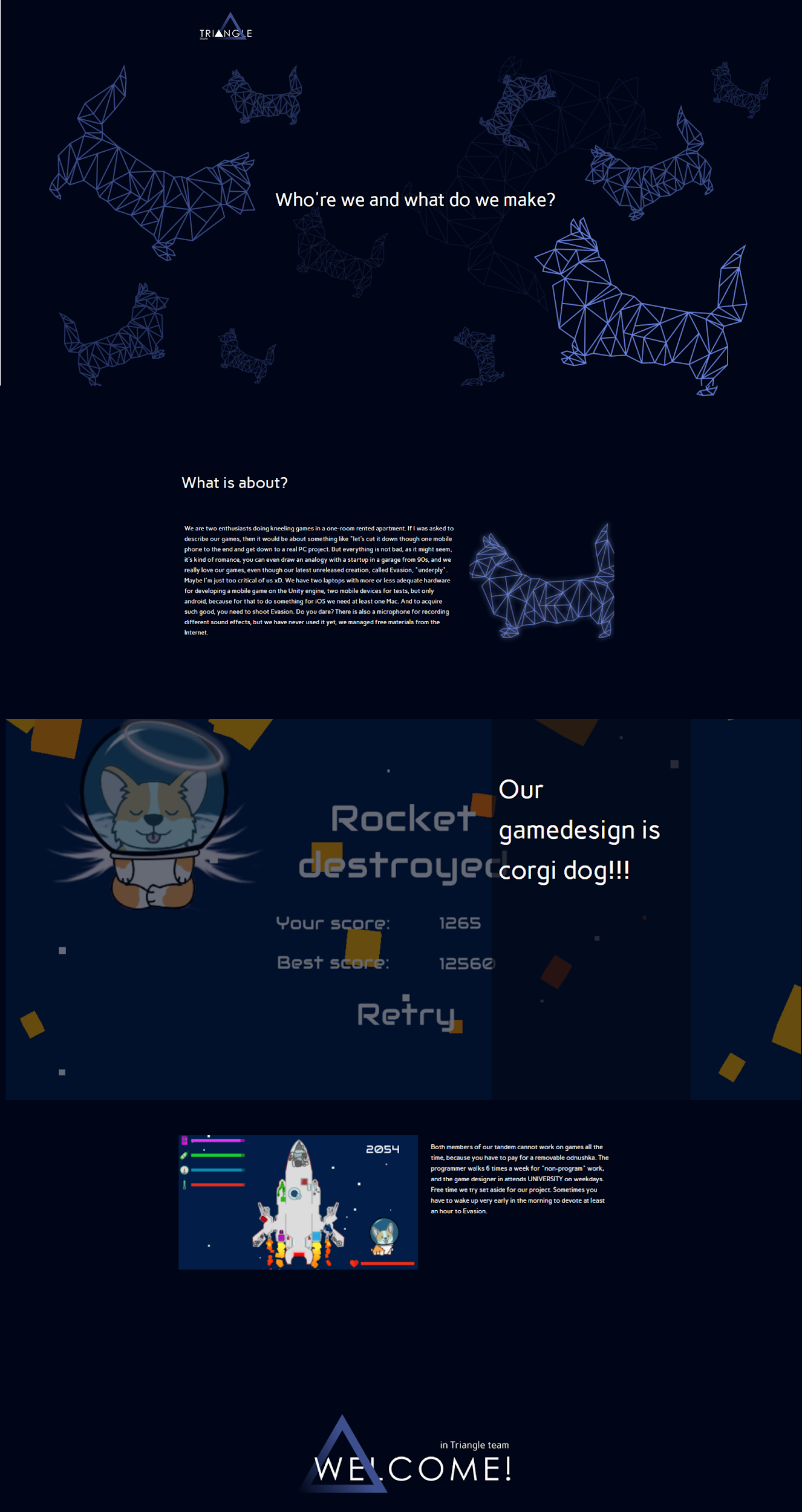
* приветственный блок с логотипом компании;
* блок новостей, оформленных в виде слайдера;
* карточки с играми;
* аккордеон с информацией об участниках команды;
* и форма обратной связи.

На главной странице имеется в качестве навигационного меню список с якорями по перемещению по странице на соответствующие блоки. Вариант дизайна списка навигации представлен на рисунке 2.6.



*Рисунок 2.6 – «Навигационное меню»*

Кроме главной страницы необходимо было разработать дизайн новостной страницы. Эта страница должна быть универсальна для всех новостных статей. Дизайн главной страницы представлен на рисунке 2.7.



*Рисунок 2.7 – «Новостная страница сайта»*

В результате был разработан дизайн сайта-лендинга с элементами витрины, определены стиль и цвета бренда, а также разработан логотип и символ команды разработчиков.

**2.2 Разработка клиентской части сайта**

Каждый раздел представляет собой компонент, состоящий из сверстанного шаблона и скрипта, написанного на Vue.js. Главная страница состоит из родительского компонента, включающего несколько дочерних компонентов, так как главная страница состоит из подразделов. В курсовой проект фреймворк Vue.js включался посредством скачивания файлов. При этом в созданном скрипте необходимо прописать подключение (листинг 3.1). Внедрение библиотеки BootstrapVue для верстки происходило включением в проект с помощью Node.js (листинг 3.2).

*Листинг 3.1 – Подключение Vue.js в главном файле*

<script src=”external-libs/vue.js”></script>

*Листинг 3.2 – Подключение BootstrapVue*

npm install vue bootstrap-vue bootstrap

BootstrapVue обеспечила быструю адаптивность элементов, а тажке 12-столбцовую сетку для структурирования. Структура главной страницы представлена в листинге 3.3. С помощью директив задавались параметры фильтрации, названий столбцов, данных для вывода и другое.

*Листинг 3.3 – Шаблон главной страницы с дочерними компонентами*

<div id='home' class="main-block">

<img src="@/assets/main\_logo.svg">

<h4 >Good solutions for your entertainment!</h4>

<b-button class="primary-button" @click.prevent='scrollTo("team")'>

More about team</b-button>

</div>

<news-block id='news' :items="news"></news-block>

<games-block id='games' :games="games"></games-block>

<team-block id='team' :members="members"></team-block>

<reviews-block id='form'></reviews-block>

<div class="sec-block footer">

<img src="@/assets/footer\_logo.svg"></div>

Для получения информации для каждого из дочерних элементов из базы данных: статей, проектов, информации об участниках команды и другая – в компонентах были созданы методы, в которых происходит вызов запроса на серверную часть. Пример метода для получения проектов на фронте представлен в листинге 3.4.

*Листинг 3.4 – Метод getGames() для получения информации об играх*

getGames(){

GameDataService.getAll()

.then(response =>{ this.games=response.data;})

.catch(e =>{console.log(e);});},

Как видно, в методе происходит обращение к сервису GameDataService, который отвечает непосредственно за передачу запроса на бэкенд. В листинге 3.5 и 3.6 представлен код сервиса с запросами на бэкенда, а также адрес, по которому необходимо обращаться.

*Листинг 3.5 – Сервис для запросов на серверную часть сайта*

import http from "../http-common";

class GameDataService {

getAll() {

return http.get("/games"); }

create(data) {

return http.post("/games", data) }

update(id, data) {

return http.put(`/games/${id}`, data);}}

export default new GameDataService();

*Листинг 3.6 – Файл объявления адреса бэкенда*

import axios from "axios";

export default axios.create({

baseURL: "http://localhost:8082/api",

headers: {"Accept": "\*",

"Access-Control-Allow-Origin": "\*",

"Access-Control-Allow-Methods":

"GET, POST, PATCH, PUT, DELETE, OPTIONS",

"Access-Control-Allow-Headers":

"Access-Control-Allow-Headers, Authorization, Content-Type, Content-Range, Content-Disposition, Content-Description, sessionId, Origin, Accept, X-Requested-With, Access-Control-Request-Method, Access-Control-Request-Headers",

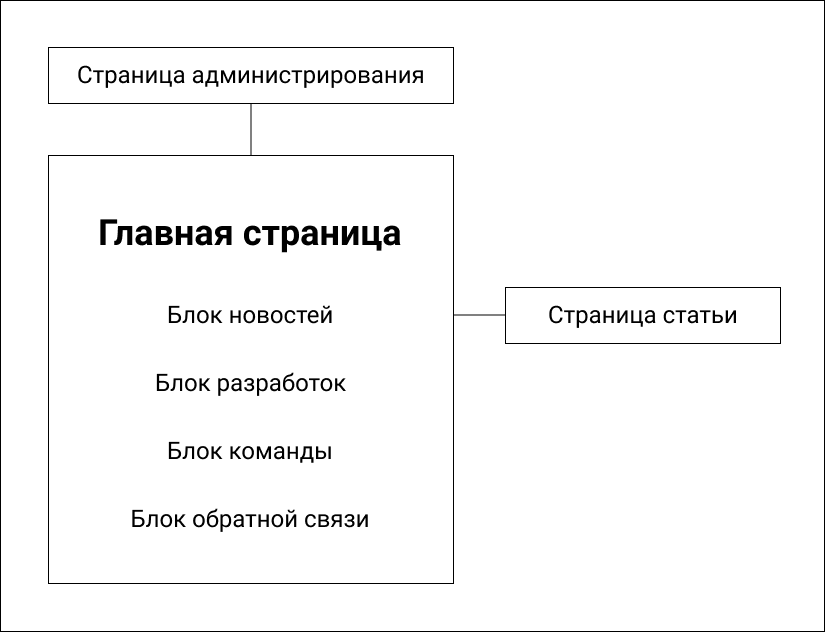
"Content-type": "application/json" }});

В этом файле также указываются заголовки, которые будут разрешены сервисом для манипуляций с информацией, а также разрешенные стандартные методы Get, Post, Patch, Put, Delete и Options.

**ГЛАВА 3. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ САЙТА**

**3.1 Проектирование структуры базы данных**

База данных для будущего сайта проектировалась, опираясь на логическую структуру, оговоренную с командой. Структура в виде схемы представлена на рисунке 2.1.

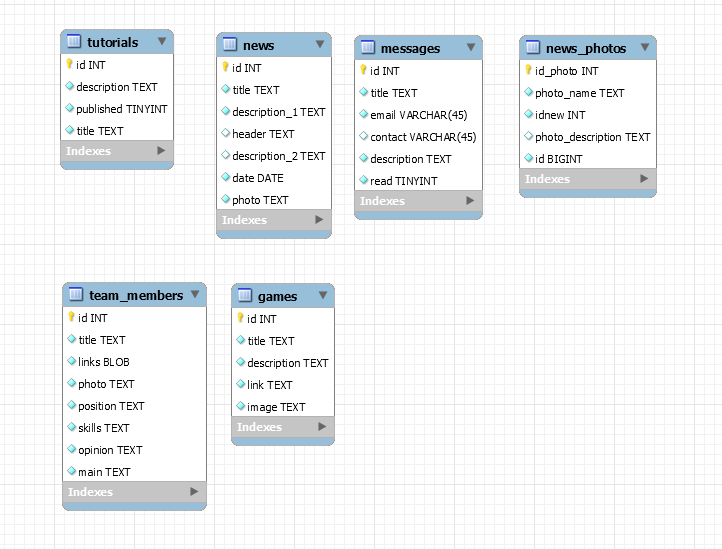


*Рисунок 2.1 – «Структура сайта гейм-разработки»*

Сайт должен размещать информацию о каждом участнике команды, карточки разрабатываемых проектов, карточки статей блога, а также возможность получать сообщения от посетителей сайта. На основе этих параметров была составлена база данных db\_games. База данных должна содержать следующие таблицы:

* News – для хранения добавляемых статей в блог с несколькими заголовками и абзацами, а также датой размещения и заглавной фотографией.
* News\_photos – таблица для хранения адресов к картинкам. Здесь также есть поле idnew, которое привязывает фотографию к определенной статье. Реализована связь от одного ко многим.
* Таблица messages для хранения сообщений, присылаемых посетителями сайта в форме обратной связи.
* Team\_members – таблицы, где хранится информация о сленах команды. Здесь описана общая информация (имя, фотография, навыки, деятельность). Кроме этого здесь хранятся в виде поля Blob данные о социальных сетях и контактах команды в формате json.
* Games – главная таблица, содержащая информацию о разработанных или еще создаваемых играх. Содержит в себе поля: название игры, краткое описание, заглавную иллюстрацию с игре и ссылку, которая ведет либо на веб-страницу, где можно поиграть в игру или же в youtube-блог, где снят обзор на игру.

Схема базы данных представлена на рисунке 2.2.

*Рисунок 2.2 – «Структура базы данных db\_games»*

База данных состоит из 6 таблиц, отвечающих за информацию об играх, членах команды, ведение блога и отправку сообщений с сайта.

**3.2 Разработка серверной части сайта**

Корректную передачу информации из базы данных обеспечивает бэкенд. Для каждого вида данных создается класс – сущность, которая содержит поля, содержащие информацию об объекте: игре, статье, сотруднике и тому подобное. Пример класса представлен в листинге 3.7.

*Листинг 3.7 – Класс «Game»*

package com.example.demo.model;

import javax.persistence.\*;

@Entity

@Table(name = "games")

public class Game {

@Id @GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(name = "title")

private String title;

@Column(name = "description")

private String description;

@Column(name = "link")

private String link;

@Column(name = "image")

private String image;

public Game() {}

public Game(String title, String description, String link, String image) {this.title=title; this.description = description;

this.link=link; this.image=image;}

public long getId() { return id; }

public String getTitle() { return title; }

public String getDescription() { return description; }

public String getImage() { return image; }

public String getLink() { return link; }

public void setId(long id) {this.id = id; }

public void setTitle(String title) { this.title = title;}

public void setDescription(String description) {

this.description = description; }

public void setImage(String image) {this.image = image;}

public void setLink(String link) { this.link = link; } }

Класс содержит следующие обязательные элементы: поля сущности, конструктор без параметров, конструктор с параметрами, необходимый для добавления в базу новых данных, сеттеры – методы для заполнения полей, геттеры – методы для чтения информации из полей.

После создания класса необходимо написать интерфейс репозитория для взаимодействия со списком объектов из базы данных. Это поможет не создавать отдельно универсальные методы такие, как поиск, сохранение, получение общего количества элементов, поиск по определенному элементу, так как используется интерфейс фреймворка Spring Data JpaRepository, предоставляющий набор стандартных методов для работы с базой данных. Реализация интерфейса представлена в листинге 3.8

*Листинг 3.8 – Интерфейс GameRepository*

package com.example.demo.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.model.Game;

public interface GameRepository extends JpaRepository<Game, Long> { List<Game> findByTitleContaining(String title); }

Наконец, создается контроллер специально для модели данных, где и предоставляется API для создания, удаления, извлечения, обновления и поиска игр, новостей, сообщений и других сущностей. Листинг контроллера для сообщений представлен в листинге 3.9.

*Листинг 3.9 – Контроллер сообщений MessageController*

package com.example.demo.controller;

import java.util.ArrayList;

import java.util.List;

///

import com.example.demo.model.Message;

import com.example.demo.repository.MessageRepository;

@CrossOrigin(origins = "http://localhost:8083/")

@RestController

@RequestMapping("/api")

public class MessageController {

@Autowired

MessageRepository messageRepository;

@GetMapping("/messages")

public ResponseEntity<List<Message>> getAllMessages(@RequestParam(required = false) String title) { try

{ List<Message> messages = new ArrayList<Message>();

if (title == null)

messageRepository.findAll().forEach(messages::add);

else

messageRepository.findByTitleContaining(title).

forEach(messages::add);

if (messages.isEmpty()) {

return new ResponseEntity<>(HttpStatus.NO\_CONTENT); }

return new ResponseEntity<>(messages, HttpStatus.OK);

} catch (Exception e)

*Продолжение листинга 3.9*

{ return new ResponseEntity<>(null, HttpStatus.INTERNAL\_SERVER\_ERROR);} }

@PostMapping("/messages")

public ResponseEntity<Message>

createTutorial(@RequestBody Message message) { try {

Message \_message = messageRepository

.save(new Message(message.getTitle(), message.getEmail(), message.getContact(), message.getDescription(), false));

return new ResponseEntity<>(\_message, HttpStatus.CREATED);}

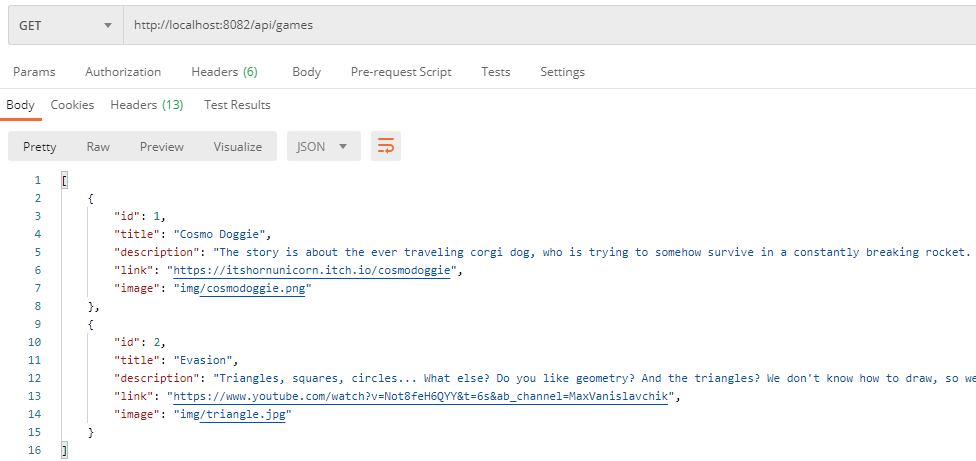
catch (Exception e)

{return new ResponseEntity<>(null, HttpStatus.INTERNAL\_SERVER\_ERROR); }} }

Стоит пояснить значение некоторых аннотаций, используемых при написании контроллера:

1. @CrossOriginпредназначен для настройки разрешенных источников.
2. @RestControllerаннотация используется для определения контроллера и для указания того, что возвращаемое значение методов должно быть привязано к телу веб-ответа.
3. @RequestMapping("/api")объявляет, что все URL-адреса Api в контроллере будут начинаться с /api.

Данные файлы полностью описывают модель и обеспечивают получение информации из базы данных. Пример работы запроса с помощью программы Postman представлен на рисунке 3.5.



*Рисунок 3.5 – «Пример REST API»*

В проекте использовалась база данных MySQL. В файле конфигураций бэкенда необходимо было прописать путь к базе данных, порт, на котором она располагается, а также логин и пароль базы данных. Код файла конфигурации представлен на рисунке 3.10.

*Листинг 3.10 – Файл application.properties*

spring.datasource.url= jdbc:mysql://localhost:3306/gamedb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC&useSSL=false

spring.datasource.username= root

spring.datasource.password= root server.port=8082

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.hibernate.ddl-auto=update

Связь базы данных с бэкендом обеспечена и работает корректно. Осталось обеспечить корректную передачу данных с бэкенда на клиентскую часть. При этом могут возникнуть проблемы с CORS (Cross-Origin Resource Sharing) – механизм для возможности получать разрешения на доступ к выбранным ресурсам с сервера на домене. Поэтому необходимо добавить файл CORSFilter.java, в котором будут прописаны разрешенные заголовки и методы, разрешенные для передачи на фронтенд. Фрагмент файла представлен в листинге 3.11.

*Листинг 3.11 – Файл CORSFilter.java*

response.setHeader("Access-Control-Allow-Origin", "\*");

response.setHeader("Access-Control-Allow-Credentials", "true");

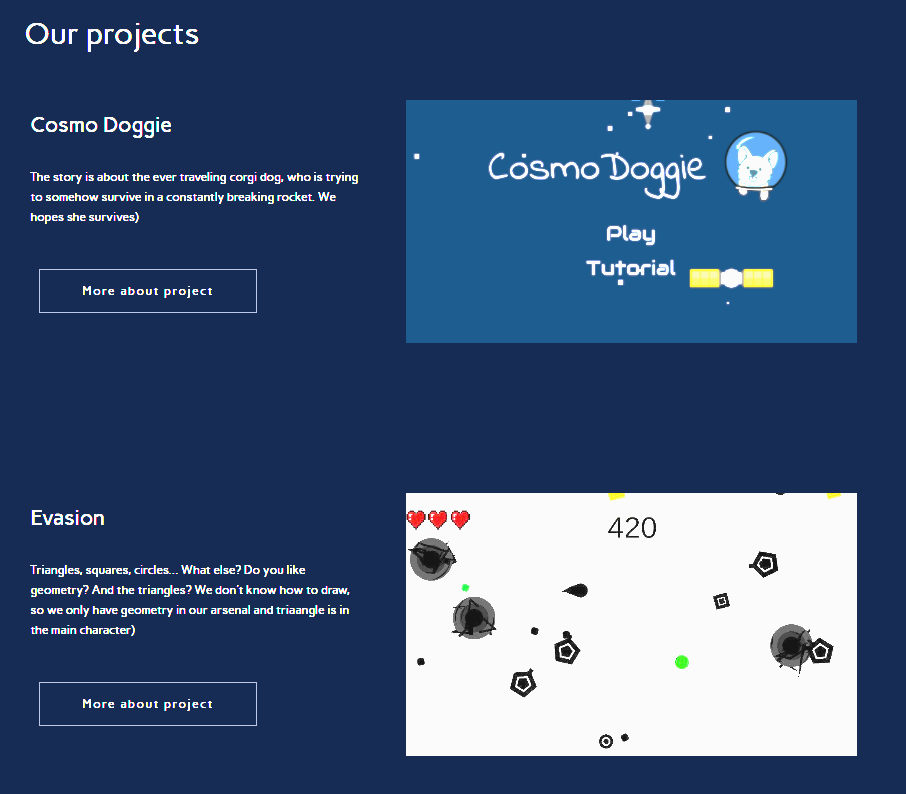
response.setHeader("Access-Control-Allow-Methods", "GET,HEAD,OPTIONS,POST,PUT");

response.setHeader("Access-Control-Allow-Headers", "Access-Control-Allow-Headers, Access-Control-Allow-Methods, Authorization, Content-Type, Content-Range, Content-Disposition, Content-Description, sessionId, Origin, Accept, X-Requested-With, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Origin");

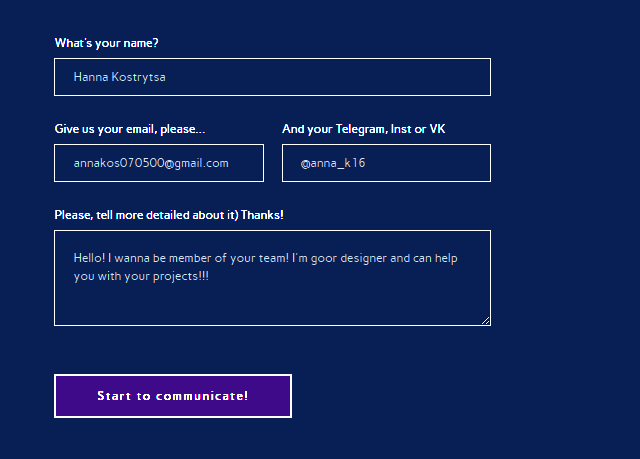
. Всего при разработке сайта были созданы и описаны следующие модели:

* Модель данных Games для отображения информации о разрабатываемых играх на главной странице сайта. Здесь реализованы методы получения информации из базы данных, а также добавления новых проектов.
* Модель данных News, которая используется для ведения командой блога. Здесь реализованы методы добавления новой статьи, получения полной статьи по id, получения всего списка статей из базы данных.
* Модель Messages, служащая для обратной связи посетителей сайта с командой. Здесь основными методами являются добавление сообщений в базу данных, а также получение всего списка сообщений в админпанели.
* Members – модель данных для отображения информации о команде. Здесь также реализованы методы получения всего списка участников команды, а также добавление нового участника команды.

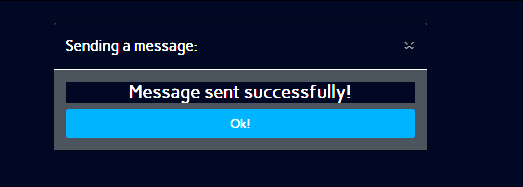
Результат получения данных о проектах представлена на рисунке 3.6. Успешная отправка сообщения демонстрируется на рисунках 3.7 и 3.8.



*Рисунок 3.6 – «Вывод проектов на сайте из базы данных»*

****

*Рисунок 3.7 – «Заполнение формы обратной связи»*

****

*Рисунок 3.8 – «Сообщение об успешной отправке сообщения»*

В результате были разработаны страницы сайта: главная страница, страницы проекта, несколько блоков для отправки информации и подтверждающий сообщений, а также страница администратора.

**ЗАКЛЮЧЕНИЕ**

В результате разработки был создан сайт-визитка команды гейм-разработчиков, которая позволяет быстро рассказать о разрабатываемых играх, об уже выпущенных играх, а также о составе команды и о инструментах, которыми пользуются члены команды при разработке. Сайт используется для привлечения заинтересованных в гейм-разработке людей.

После определения списка используемых технологий проводилось изучение документации по работе с фреймворком Vue.js, библиотеками Vue-router.js, BootstrapVue, документации по фреймворку SpringBoot и его работе с Vue.js и базой данных MySQL.

В ходе выполнения проекта были созданы следующие страницы:

* главная страница с разделами:
* раздел «Блог»;
* раздел «Проекты»;
* раздел «Команда»;
* раздел «Обратная связь»;
* страница статьи;
* панель администратора.

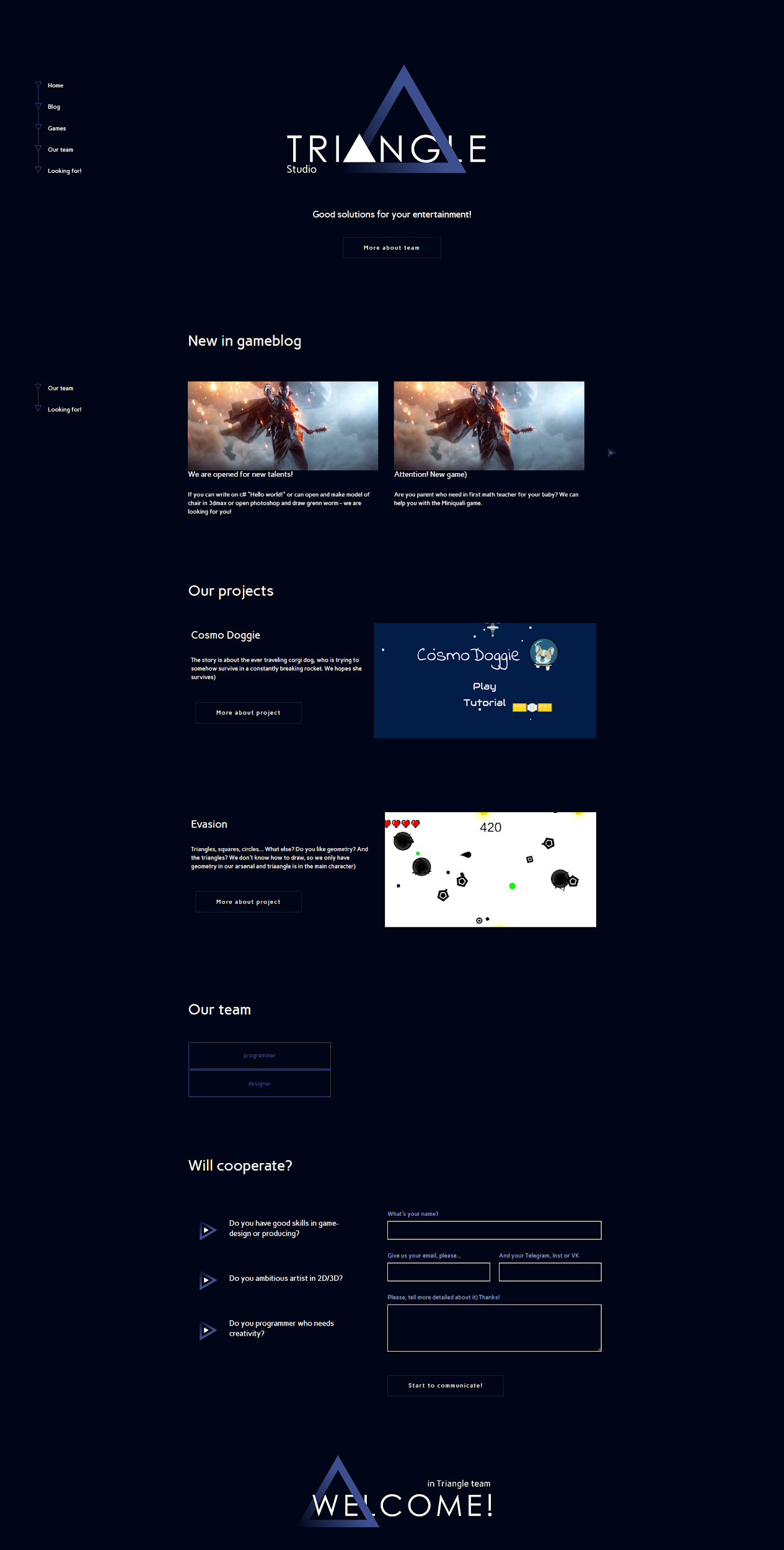
На сайте демонстрируются разработки команды; реализовано ведение блога; презентуются участники команды; обеспечена обратная связь с посетителями сайта.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

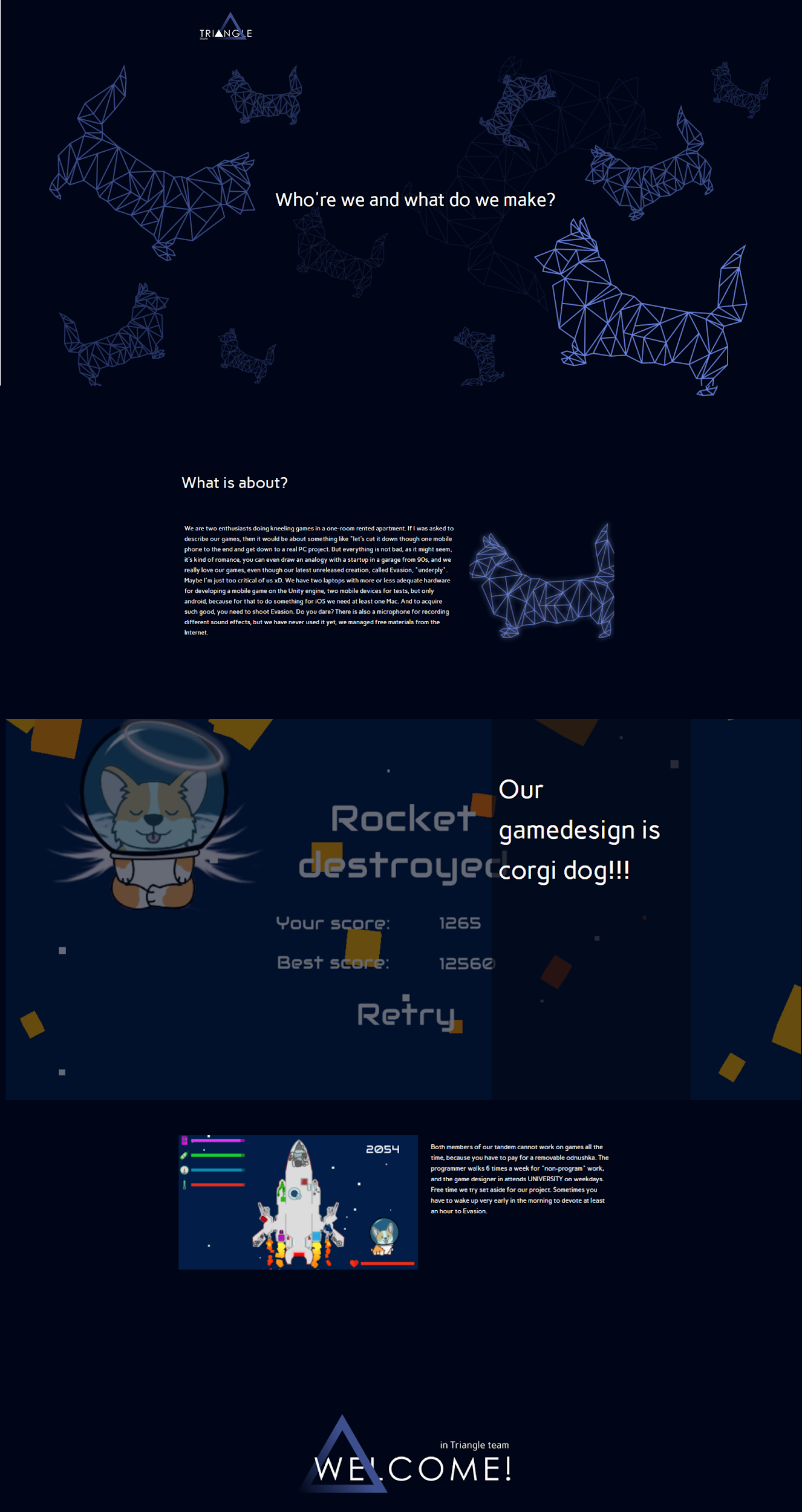
1. Game studio «Amplitude Studios» [Electronic resource] *–* Mode of access: https://www.amplitude-studios.com/#Home. – Date of access: 09.10.2020.
2. Game corporation «CPP Games» [Electronic resource] *–* Mode of access: https://www.ccpgames.com/ – Date of access: 09.10.2020.
3. Getting started with BootstrapVue [Electronic resource] *–* Mode of access: <https://bootstrap-vue.org/docs>. *–* Date of access: 25.10.2020.
4. Э. Ханчетт. Vue.js в действии / Э.Ханчетт, Б. Листуон; пер. изд. «Питер». *–* Нью-Йорк: Manning Publications Co, 2020. *–* 24 c.
5. Introduction. What is Vue.js? Documentation. [Electronic resource] *–* Mode of access: <https://vuejs.org/v2/guide/>. *–* Date of access: 24.10.2020.
6. Библиотека маршрутизации Vue Router [Электронный ресурс] *–* Режим доступа: <https://router.vuejs.org/ru/> *–* Дата доступа: 25.10.2020.
7. Spring Boot – Краткое руководство [Электронный ресурс] – Режим доступа: <https://coderlessons.com/tutorials/java-tekhnologii/learn-spring-boot/> – Дата доступа: 25.10.2020.
8. Spring Boot [Electronic resource] – Mode of access: <https://spring.io/projects/spring-boot>. – Date of access: 25.10.2020

ПРИЛОЖЕНИЕ А

Дизайн сайта «Triangle»



*Рисунок А1 – Главная страница сайта*



*Рисунок А2 – Новостная страница сайта*

ПРИЛОЖЕНИЕ Б

Код страниц сайта на Vue.js

1. Компонент главной страницы

<template>

<div><div class='navbar'>

<ul class='navbar1'><li id="first-nav" class="lll active"

@click.prevent='scrollTo("home")'><a>Home</a></li>

<li class="lll" @click.prevent='scrollTo("news")'><a>Blog</a></li>

<li class="l" @click.prevent='scrollTo("games")'><a>Games</a></li>

<li class="lll" @click.prevent='scrollTo("team")'><a>Our team</a></li>

<li class="lll" @click.prevent='scrollTo("form")'><a>Looking for!</a></li></ul>

</div><div id='home' class="main-block">

<img src="@/assets/main\_logo.svg">

<h4 >Good solutions for your entertainment!</h4>

<b-button class="primary-button" @click.prevent='scrollTo("team")'>More about team</b-button></div>

<news-block id='news' :items="news"></news-block>

<games-block id='games' :games="games"></games-block>

<team-block id='team' :members="members"></team-block>

<reviews-block id='form'></reviews-block>

<div class="sec-block footer">

<img src="@/assets/footer\_logo.svg"></div> </div></template>

<script>

import GameDataService from "../../services/GameDataService";

import MemberDataService from "../../services/MemberDataService";

import NewsDataService from"../../services/NewsDataService"

import newsBlock from "./newsBlock.vue"

import gamesBlock from "./gamesBlock.vue"

import teamBlock from "./teamBlock.vue"

import reviewsBlock from "./reviewsBlock"

export default {

name: "mainPage",

components:{ newsBlock, gamesBlock, teamBlock, reviewsBlock },

data(){return{ games:[], members: [], news:[],} },

created(){ this.getGames(); this.getMembers(); this.getNews();},

methods:{ scrollTo(idLink){

let scrollEl = document.getElementById(idLink);

scrollEl.scrollIntoView({block: 'start', behavior: 'smooth'}); },

getGames(){ GameDataService.getAll()

.then(response =>{ this.games=response.data; })

.catch(e =>{ console.log(e); });},

getNews(){NewsDataService.getAll()

.then(response =>{

this.news=response.data; })

.catch(e =>{ console.log(e); }); },

getMembers(){let cont=this; MemberDataService.getAll()

.then(response =>{ cont.members=response.data;})

.catch(e =>{console.log(e);});},},}</script>

1. Компонент страницы статьи

<template><div><div class="header">

<a href="/" ><img style="width: 14%;" src="../assets/main\_logo.svg"/></a></div>

<div class="new\_photo" v-if="photos" style=" width: 100%;">

<img style="width: 180%; margin-left: -40%;" :src="photos[0].photo\_name"/> <h1 style="font-size: 3rem; width: 90%; ">{{currentNew.title}}</h1></div>

<div class="sec-block">

<h1 >What is about?</h1>

<div class="game-card"><div class="game-info">

<p style="font-size: 16px;">{{currentNew.description\_1}}</p></div>

<div class="game-image">

<img :src="photos[1].photo\_name"/> </div></div>

<div class="new\_photo" v-if="photos" style=" width: 100%; height:60rem;"> <img style="width: 180%; margin-left: -40%; opacity: 40%; height:100%; object-fit: cover;" :src="photos[2].photo\_name"/>

<div class="news\_h3"><h2 class="news\_title">{{currentNew.header}}</h2></div></div>

<div class="game-card" style="margin-top: 160px;">

<div class="game-image"> <img :src="photos[3].photo\_name"/> </div>

<div class="game-info" style="padding-left:24px;">

<p style="font-size: 16px;">{{currentNew.description\_2}}</p> </div></div> <div class="sec-block footer">

<img src="@/assets/footer\_logo.svg"></div></div></div> </template>

<script> import NewsDataService from"../services/NewsDataService"

export default {

name: "newPage",

props:{publication: Object},

data(){return{currentNew: null,photos: [], }},

created(){this.currentNew = this.$router.currentRoute.query.item;

this.photos=this.getImages(this.currentNew.id);},

mounted(){console.log(this.photos);this.$router.currentRoute.query.item);},

methods:{getImages(id){NewsDataService.getPhotos(id)

.then(response =>{this.photos=response.data;})

.catch(e =>{console.log(e);}); },}}</script>

1. Файлы-сервисы для запросов в базу данных

import http from "../http-common";

class GameDataService {

getAll() {return http.get("/games");}}

export default new GameDataService();

import http from "../http-common";

class MemberDataService {return http.get("/members"); }}

export default new MemberDataService()

import http from "../http-common";

class MessageDataService {

getAll() {return http.get("/messages");}

create(data) {return http.post("/messages", data); }}

export default new MessageDataService();

import http from "../http-common";

class NewsDataService {

getAll() { return http.get("/news"); }

getPhotos(id) { return http.get(`/photos/${id}`);}

create(data) {return http.post("/news", data); }}

export default new NewsDataService();

import http from "../http-common";

class TutorialDataService {

getAll() {return http.get("/tutorials");}

get(id) { return http.get(`/tutorials/${id}`); }

create(data) {return http.post("/tutorials", data); }

update(id, data) { return http.put(`/tutorials/${id}`, data);}

delete(id) {return http.delete(`/tutorials/${id}`);

deleteAll() {return http.delete(`/tutorials`); }

findByTitle(title) { return http.get(`/tutorials?title=${title}`);}}

export default new TutorialDataService();

ПРИЛОЖЕНИЕ В

Java-код некоторых моделей данных и контроллеров

1. Модель Messages

package com.example.demo.model;

import javax.persistence.\*;

@Entity

@Table(name = "messages")

public class Message {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(name = "`title`")

private String title;

@Column(name = "email")

private String email;

@Column(name = "contact")

private String contact;

@Column(name = "description")

private String description;

@Column(name="`read`")

private Boolean read;

Message(){}

public Message(String title, String email, String description, Boolean read){

this.title=title; this.email=email;

this.description=description; this.read=read; }

public Message(String title, String email, String contact, String description, Boolean read){

this.title=title; this.email=email;

this.contact=contact; this.description=description;

this.read=read; }

public long getId() { return id; }

public String getTitle() { return title; }

public String getEmail() { return email; }

public String getContact() { return contact;}

public String getDescription() { return description; }

public Boolean getRead() { return read;}

public void setId(long id) { this.id = id; }

public void setTitle(String title) { this.title = title; }

public void setEmail(String email) { this.email = email;}

public void setContact(String contact) {

this.contact = contact; }

public void setDescription(String description) {

this.description = description; }

public void setRead(Boolean read) { this.read = read;}}

package com.example.demo.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.model.Message;

public interface MessageRepository extends JpaRepository<Message, Long> {List<Message> findByRead(boolean read);

List<Message> findByTitleContaining(String title);}

package com.example.demo.controller;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.CrossOrigin;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import com.example.demo.model.Message;

import com.example.demo.repository.MessageRepository;

@CrossOrigin(origins = "http://localhost:8083/")

@RestController

@RequestMapping("/api")

public class MessageController {

@Autowired MessageRepository messageRepository;

@GetMapping("/messages")

public ResponseEntity<List<Message>> getAllMessages(@RequestParam(required = false) String title) {

try { List<Message> messages = new ArrayList<Message>();

if (title == null) messageRepository.findAll().forEach(messages::add);

else messageRepository.findByTitleContaining(title).forEach(messages::add);

if (messages.isEmpty()) {

return new ResponseEntity<>(HttpStatus.NO\_CONTENT); }

return new ResponseEntity<>(messages, HttpStatus.OK);

} catch (Exception e) {

return new ResponseEntity<>(null, HttpStatus.INTERNAL\_SERVER\_ERROR);} }

@PostMapping("/messages")

public ResponseEntity<Message> createTutorial(@RequestBody Message message) {

try { Message \_message = messageRepository

.save(new Message(message.getTitle(), message.getEmail(), message.getContact(), message.getDescription(), false));

return new ResponseEntity<>(\_message, HttpStatus.CREATED);

} catch (Exception e) {

return new ResponseEntity<>(null, HttpStatus.INTERNAL\_SERVER\_ERROR);} } }