**Реферат**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

Реферат

Лит.

Листов

1

*74417039, 2022*

Пояснительная записка содержит 104 страницы, 50 рисунков, 20 таблиц,   
13 источников, 10 приложений.

ВЕБ-ПРИЛОЖЕНИЕ «ТУРИСТИЧЕСКОЕ АГЕТСТВО», JAVASCRIPT, NODE.JS, MONGODB, REACT

Основной целью дипломного проекта является разработка веб-приложения «Туристическое агентство».

Пояснительная записка состоит из введения, шести разделов и заключения.

Во введении представлена общая информация о текущем состоянии в сфере рассматриваемой проблемы.

В первом разделе представлена общая теоретическая информация по теме, обзор существующих аналогов разрабатываемой системы и постановка задачи.

Во втором разделе произведено проектирование веб-приложения.

В третьем разделе показан процесс разработки веб-приложения.

В четвертом разделе показан процесс тестирования разработанного веб-приложения.

В пятом разделе представлена методика использования разработанного веб-приложения.

В шестом разделе представлены результаты себестоимости и отпускной цены разработанного веб-приложения.

В заключении представлены итоги дипломного проекта и задачи, которые были решены в ходе разработки веб-приложения.

**Abstract**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

.

Утв.

Пацей Н. В.

Abstract

Лит.

Листов

1

*74417039, 2022*

Explanatory letter of degree project is performed in the content of 104 pages, 50 images, 20 tables, 13 literary sources, 10 annexes.

WEB APP TRABEL AGENCY, JAVASCRIPT, NODE.JS, MONGODB, REACT

Explanatory letter of degree project contains indroduction, 6 sections, summary.

Degree project goal is to build a web application «Travel Agency».

The first section is a problem solving and analytical literature review on the topic of the course project

The second section describes the system design process and technology used during the web-application.

The third section describes the process of development, the principles of the functional-nation and purpose of the created components of the web-application.

In the fourth section the testing is described, which helps to understand the interface of the web-application in details and test results of the work done.

The fifth section describes how to install and use the web-application.

The sixth section presents definitions of the cost and selling price of the web-application.

In the conclusion the results of the degree project and tasks that were solved during the development of the web-application are presented.

**Содержание**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

.

Утв.

Пацей Н. В.

Содержание

Лит.

Листов

1

*74417039, 2022*

[Введение 7](#_Toc104887808)

[1 Аналитический обзор литературы и постановка задачи 8](#_Toc104887809)

[1.1 Основные понятия 8](#_Toc104887810)

[1.2 Обзор аналогов 8](#_Toc104887811)

[1.2.1 Туристическое агентство *Tez-tour* 8](#_Toc104887812)

[1.2.2 Туристическое агентство *Tury.by* 11](#_Toc104887813)

[1.2.3 Туристическое агентство *Tursales.by* 13](#_Toc104887814)

[1.2.4 Туристическое агентство *Peopletravel.by* 15](#_Toc104887815)

[1.3 Постановка задачи 17](#_Toc104887816)

[1.4 Вывод по разделу 18](#_Toc104887817)

[2 Проектирование веб-приложения 19](#_Toc104887818)

[2.1 Используемые средства разработки 19](#_Toc104887819)

[2.2 Архитектура веб-приложения 19](#_Toc104887820)

[2.3 Выбор средств реализации программных средств 21](#_Toc104887821)

[2.4 Диаграмма вариантов использования 23](#_Toc104887822)

[2.5 Проектирование базы данных 24](#_Toc104887823)

[2.6 Выводы по разделу 30](#_Toc104887824)

[3 Разработка веб-приложения 31](#_Toc104887825)

[3.1 Разработка серверной части 31](#_Toc104887826)

[3.2 Разработка клиентской части 36](#_Toc104887827)

[3.3 Выводы по разделу 41](#_Toc104887828)

[4 Тестирование веб-приложения 42](#_Toc104887829)

[4.1 Тестирование основного функционала веб-приложения 42](#_Toc104887830)

[4.2 Тестирование безопасности базы данных и веб-приложения 44](#_Toc104887831)

[4.3 Выводы по разделу 47](#_Toc104887832)

[5 Методика установки и использования 48](#_Toc104887833)

[5.1 Развёртывание системы 48](#_Toc104887834)

[5.2 Руководство пользователя 48](#_Toc104887835)

[5.2.1 Иерархическая система ролей 48](#_Toc104887836)

[5.2.2 Основные возможности роли «Гость» 48](#_Toc104887837)

[5.2.3 Основные возможность роли «Пользователь» 53](#_Toc104887838)

[5.2.4 Основные возможности роли «Администратор» 56](#_Toc104887839)

[6 Технико-экономическое обоснование дипломного проекта 60](#_Toc104887840)

[6.1 Общая характеристика разрабатываемого программного средства 60](#_Toc104887841)

[6.2 Исходные данные для проведения расчётов 60](#_Toc104887842)

[6.3 Методика обоснования цены 61](#_Toc104887843)

[6.3.1 Определение объёма программного средства 62](#_Toc104887844)

[6.3.2 Основная заработная плата 64](#_Toc104887845)

[6.3.3 Дополнительная заработная плата 64](#_Toc104887846)

[6.3.4 Расчёт отчислений на социальные цели 65](#_Toc104887847)

[6.3.5 Расходы на материалы 65](#_Toc104887848)

[6.3.6 Расходы на оплату машинного времени 65](#_Toc104887849)

[6.3.7 Расчет прочих прямых затрат 66](#_Toc104887850)

[6.3.8 Расчет общепроизводственных и общехозяйственных расходов 66](#_Toc104887851)

[6.3.9 Расчет суммы расходов на разработку программного средства 67](#_Toc104887852)

[6.3.10 Расходы на сопровождение и адаптацию 67](#_Toc104887853)

[6.3.11 Расчет общей суммы расходов 67](#_Toc104887854)

[6.3.12 Определение цены, оценка эффективности 67](#_Toc104887855)

[6.4 Вывод по разделу 70](#_Toc104887856)

[Заключение 72](#_Toc104887857)

[Список используемых источников 75](#_Toc104887858)

[ПРИЛОЖЕНИЕ А](#_Toc104887859) [Схема архитектуры приложения 74](#_Toc104887860)

[ПРИЛОЖЕНИЕ Б](#_Toc104887861) [Диаграмма вариантов использования 75](#_Toc104887862)

[ПРИЛОЖЕНИЕ В](#_Toc104887863) [Логическая структура базы данных 76](#_Toc104887864)

[ПРИЛОЖЕНИЕ Г](#_Toc104887865) [Блок-схема алгоритма бронирования тура 77](#_Toc104887866)

[ПРИЛОЖЕНИЕ Д](#_Toc104887867) [Диаграмма классов 78](#_Toc104887868)

[ПРИЛОЖЕНИЕ Е](#_Toc104887869) [Диаграмма последовательности работы чата 79](#_Toc104887870)

[ПРИЛОЖЕНИЕ Ж](#_Toc104887871) [Скриншот работы веб-приложения 80](#_Toc104887872)

[ПРИЛОЖЕНИЕ И](#_Toc104887873) [Листинг роутера туров 81](#_Toc104887874)

[ПРИЛОЖЕНИЕ К](#_Toc104887875) [Листинг роутера панели администратора 89](#_Toc104887876)

[ПРИЛОЖЕНИЕ Л](#_Toc104887877) [Листинг *React*-компонента *Tour* 92](#_Toc104887878)

# **Введение**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

Введение

Лит.

Листов

1

*74417039, 2022*

Туризм является неотъемлемой частью жизни людей. За последние десятилетия туризм стал одним из самых динамичных и быстрорастущих секторов мировой экономики, который способствует обеспечению высокого уровня занятости, социального благосостояния и качества жизни людей.

Туристическое агентство – это фирма, занимающаяся реализацией туров населению, организуемых туроператорами, а также продажей потребителям отдельных туристических услуг (транспортных билетов, экскурсий, размещение трансферов и др.).

Персональный компьютер и Интернет, их доступность и надежность, способствуют проникновению во все сферы общества новых информационных технологий. Эти технологии обеспечивают рост производительности в сфере услуг, в частности и в туризме. Ведь туризм и информация неразделимы.

Целью проекта является разработка веб-приложения, которое поможет пользователям найти подходящий тур и забронировать на него место, а также повысит эффективность работы туристического агентства.

Для достижения цели сформулированы следующие задачи:

− выбор и обоснование средств разработки;

− проектирование базы данных;

− разработка структуры серверной части;

− разработка функций серверной части;

− проектирование веб-интерфейса сайта;

− разработка пользовательского интерфейса сайта;

− тестирование серверной части и пользовательского интерфейса.

Практическая значимость: позволяет любому пользователю в интернете найти подходящий тур и забронировать место.

# **1 Аналитический обзор литературы и постановка задачи**

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

1 Аналитический обзор литературы и постановка задачи

Лит.

Листов

11

*74417039, 2022*

В настоящее время путь турфирмы к успеху заключается в том, чтобы довести до потенциального клиента соответствующую информацию, заинтересовать его и вызвать его ответные действия. Для турагентства, веб-приложение, это быстрый выход на клиентов. Поэтому сейчас уже трудно представить эффективную работу туристической фирмы без собственного сайта. Не смотря на это, у многих турфирм, до сих пор нет собственных приложений.

* 1. **Основные понятия**

**Туристическое агентство** – **это частная компания, которая выступает в качестве посредника между своими клиентами и определенными поставщиками в категории путешествий, такими как авиакомпании, отели, круизы, среди прочего, предлагая первые лучшие условия контрактов на поездки, которые они хотят предпринять**.

Веб-приложение – это приложение, одна часть которого загружается в браузер и взаимодействует с пользователем (визуально-интерфейсная часть), а другая находится на веб-сервере и выполняет запросы, поступающие от первой, а затем возвращает ответ [1].

* 1. **Обзор аналогов**

Для составления технических требований к проекту был проведён анализ похожих сайтов. В процессе анализа были рассмотрены достоинства и недостатки найденных проектов, на основе которых разрабатывались технические требования для создаваемого веб-приложения. Ссылки на материалы, используемые в данной главе, предоставлены в списке литературы.

Из открытых источников были рассмотрены следующие веб-приложения туристических агентств:

* *Tez-tour*;
* *Tury.by*;
* *Tursales.by*;
* *Peopletravel.by.*
  + 1. **Туристическое агентство *Tez-tour***

Агентство *Tez-tour* – это международный туристический оператор, организующий туры для путешественников из России и стран СНГ.

Сайт туристического агентства *Tez-tour* предназначен для ознакомления с возможными турами, предоставляет удобный поиск, с возможность последующего бронирования [2]. Насчитывается более 25 различных направлений для путешествий.

На главной странице данного сайта размещена форма поиска тура, с возможностью указания города вылета, страны назначения, даты начала, количества ночей и количество туристов в группе. Ниже на странице находятся списки с лучшими предложениями для отдыха, текущие акции и хиты продаж. Главная страница сайта представлена на рисунке 1.1.

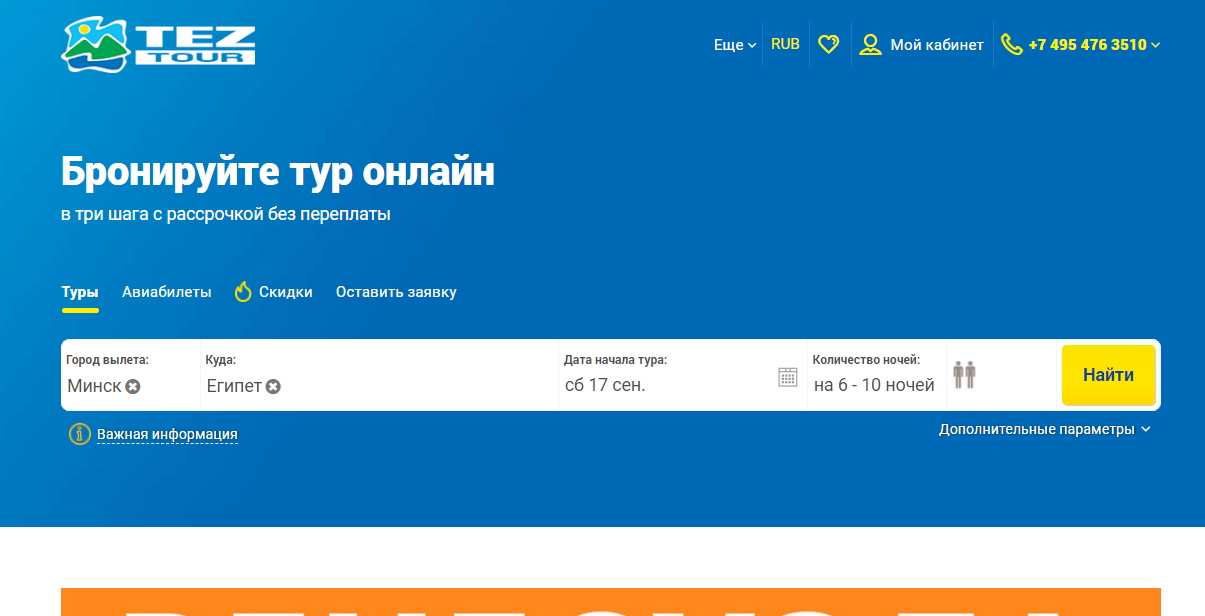


Рисунок 1.1 – Главная страница сайта *Tez-tour*

После выбора направления тура, перед пользователем появится дополнительная форма, в которой он увидит список всех доступных курортов в каждом регионе и список отелей, с возможностью выбора количества звёзд и типа питания. В качестве дополнительных параметром пользователь может указать свой бюджет и ввести промокод. Данная форма представлена на рисунке 1.2.

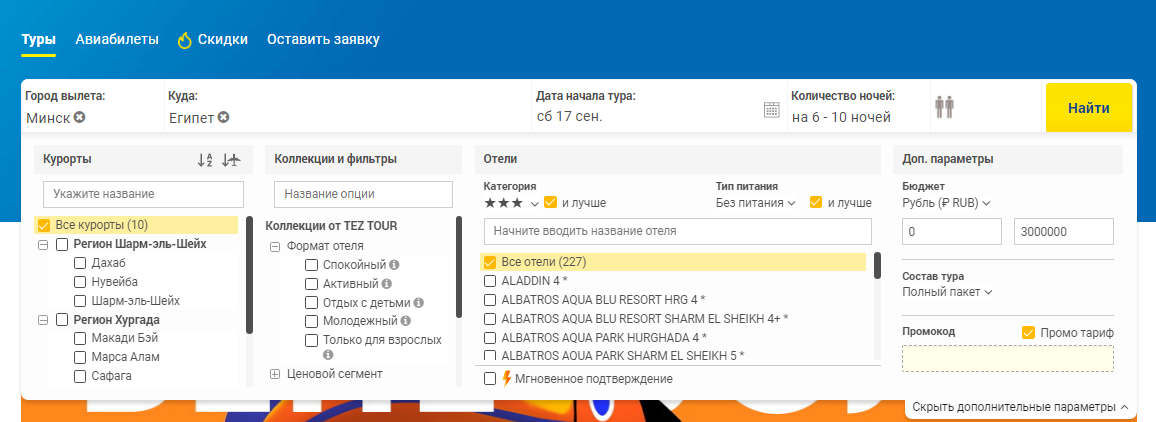


Рисунок 1.2 – Форма расширенного поиска тура

После того, как пользователь выбрал тур, приложение выполнит расчёт стоимости и предложит перейти к бронированию. Есть две опции: бронирование онлайн и в агентстве. Данная страница представлена на рисунке 1.3.



Рисунок 1.3 – Страница бронирования тура

После завершения бронирования, пользователь получит *email* письмо на почту с информацией по заказу и номеру бронирования. Отслеживать статус заказа можно в личном кабинете. Данная страница представлена на рисунке 1.4.

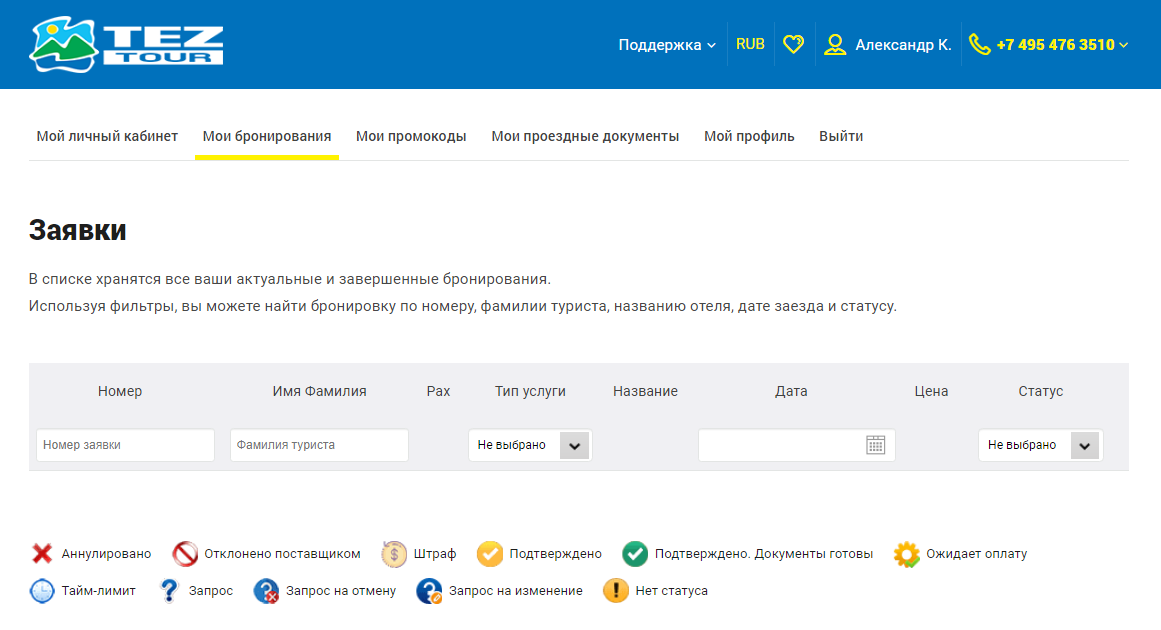


Рисунок 1.4 – Страница личного кабинета

Функционал приложения включает в себя:

* поиск туров с различными фильтрами;
* выбор конкретных курортов и отелей;
* бронирование онлайн и в агентстве;
* личный кабинет;
* промокоды.

Из недостатков стоит выделить отсутствие отзывов и плохую адаптированность под мобильные устройства.

* + 1. **Туристическое агентство *Tury.by***

Агентство *Tury.by* – данный сайт предназначен для удобного подбора туров [3]. Ассортимент туров, представленных на сайте, позволяет подобрать предложения для разных категорий туристов: тем, кто ищет наиболее экономичный вариант, и тем, кто хочет отдохнуть в дорогом отеле. Главная страница сайта с формой поиска представлена на рисунке 1.5.

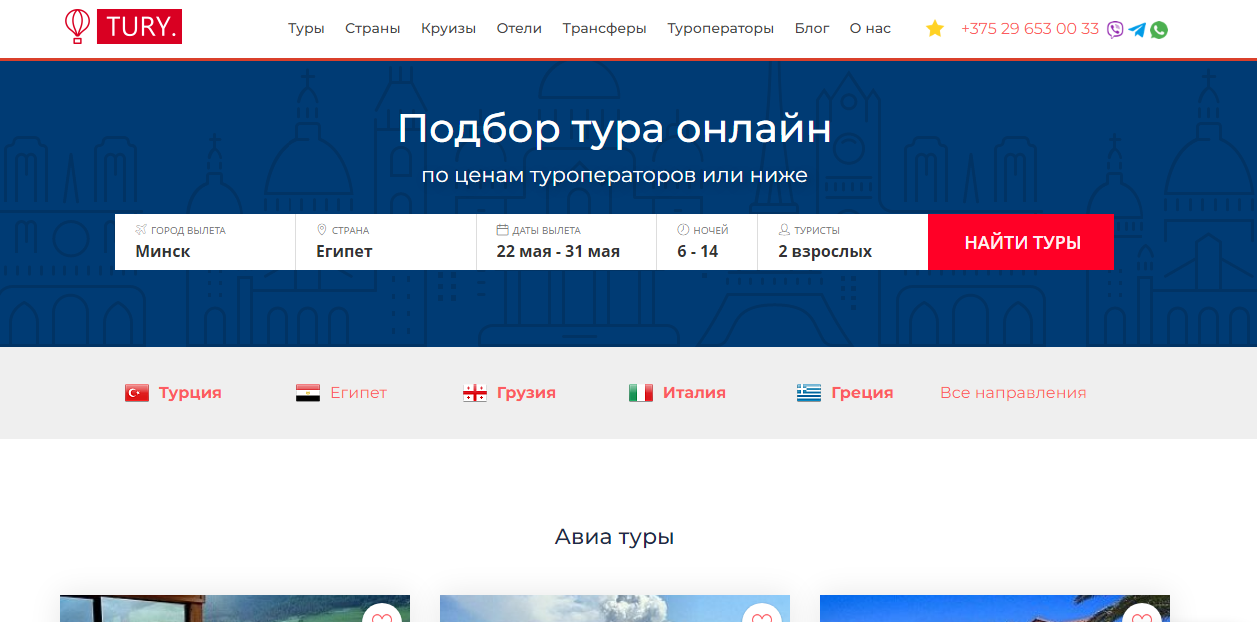


Рисунок 1.5 – Главная страница сайта *Tury.by*

В результате поиска пользователь увидит список туров и шесть фильтров в левой части приложения. Пользователь может выбрать:

* класс отеля от одной до пяти звёзд;
* тип курорта/отеля;
* питание;
* рейтинг тура;
* дополнительные услуги отеля;
* диапазон цен.

Также есть возможность выполнить сортировку по цене и посмотреть расположение отелей на карте.

Результат поиска туров представлен на рисунке 1.6.

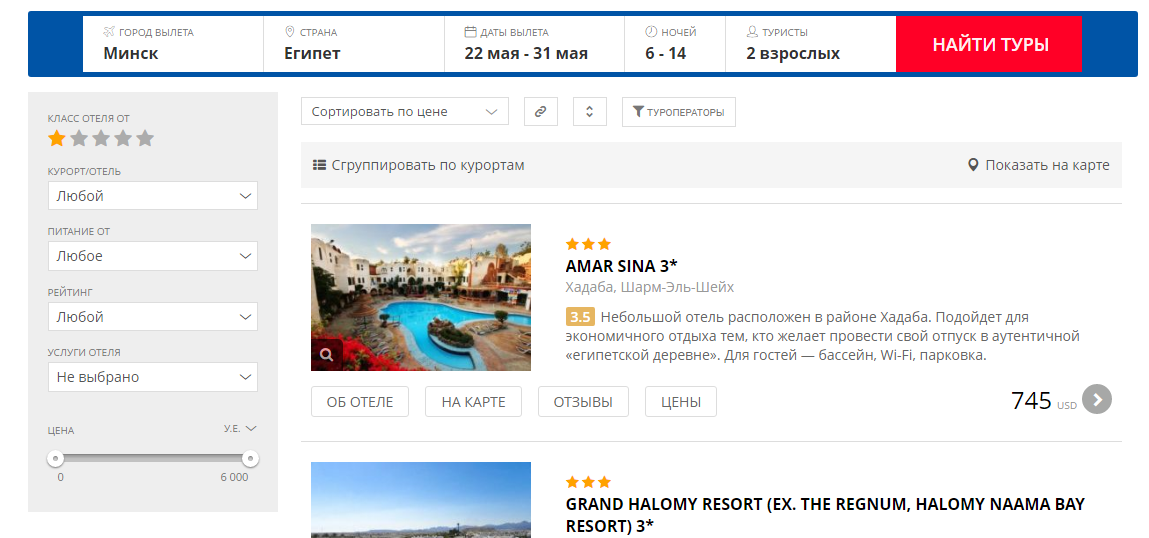


Рисунок 1.6 – Результат поиска туров сайта *Tury.by*

На странице тура пользователь может оставить заявку на тур, для получения наиболее полной информации. Данная страницы тура представлена на рисунке 1.7.

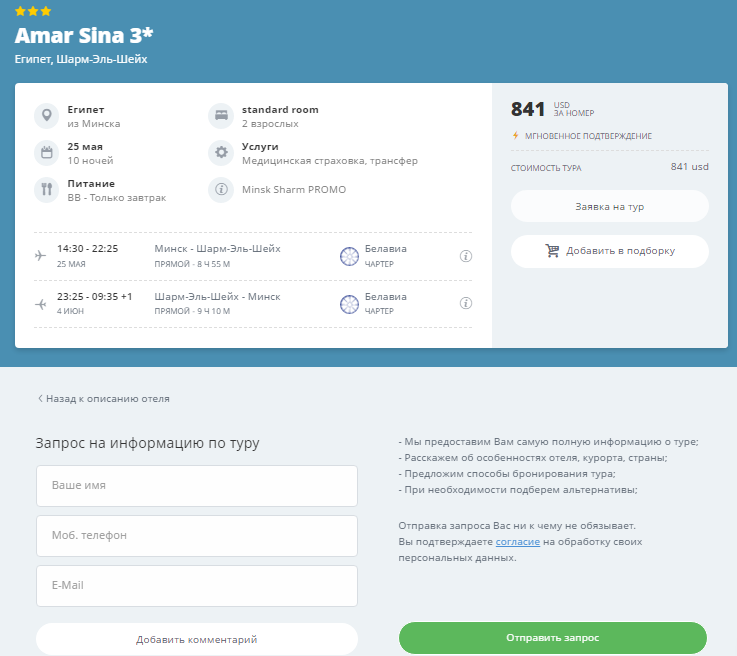


Рисунок 1.7 – Страница с запросом на информацию по туру

Данное приложение даёт следующие возможности:

* удобный интерфейс;
* отображение отелей на карте;
* чат поддержки.

В результате исследования были найдены следующие недостатки:

* нет возможности бронирования тура онлайн;
* нет возможности создать аккаунт.
  + 1. **Туристическое агентство *Tursales.by***

Агентство *Tyrsales.by* – это сайт быстроразвивающейся белорусской компании с большим выбором туров, которые включают в себя пляжный отдых, экскурсионные и горнолыжные туры, оздоровительные путёвки, а также автобусные туры по Европе. Сайт предназначен для поиска и бронирования туров [4].

Для подбора тура необходимо воспользоваться формой «Подбор тура» размещенной на главной странице сайта. Данная страница представлена на рисунке 1.8.

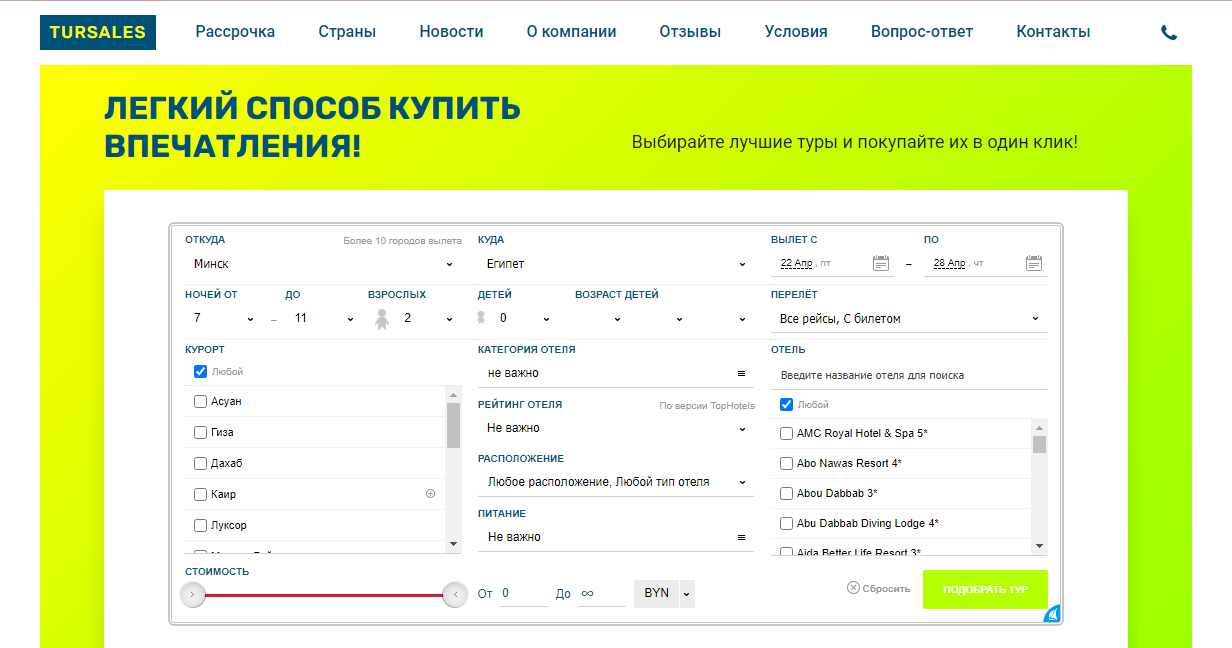


Рисунок 1.8 – Главная страница сайта *Tursales.by*

В результатах подбора тура будут отображены все туристические пакеты, в том числе и горящие, которые доступны для бронирования на сайте в зависимости от города вылета, страны, курорта, даты начала тура, количества туристов и ночей. Стоимость тура указывается за то количество туристов, которое вы указали в форме подбора.

Также можно указать тип транспорта в туре (поезд, паром, автобус, автомобиль), а также включить или исключить в стоимость тура стоимость билета.

Результаты поиска туров будут сгруппированы по отелям в зависимости от типа номера, даты начала тура и количества ночей. Данная страница представлена на рисунке 1.9.

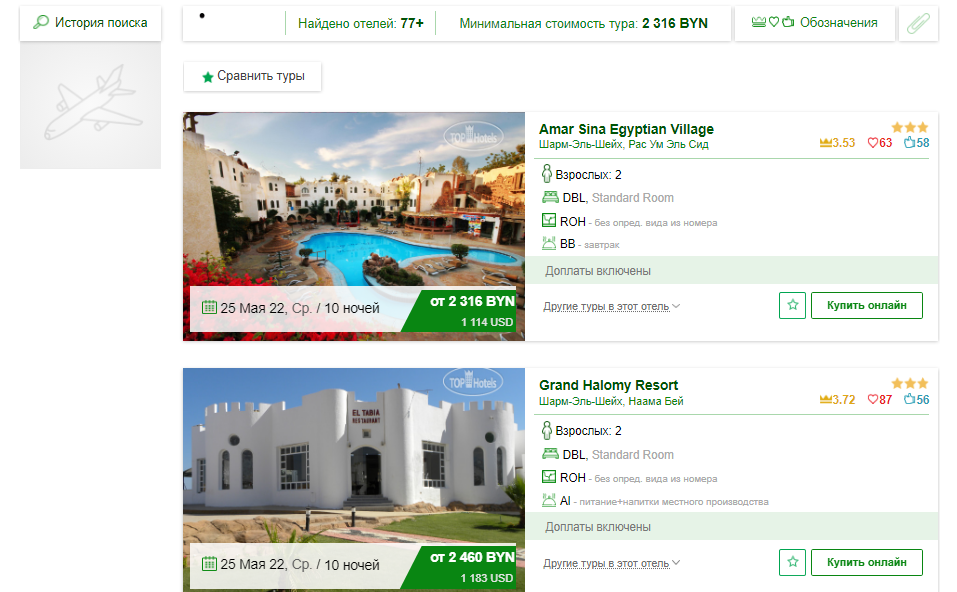


Рисунок 1.9 – Страница с результатом поиска

На странице тура есть подробная информация о отеле, отзывы пользователей, а также рейтинг и рекомендации.

После выбора тура пользователем, можно переходить к бронированию. Для этого необходимо заполнить и отправить форму заявки. Стоит отметить, что позже могут взиматься дополнительные сборы, которые не отображены в стоимости тура. Они будут добавлены менеджером при обработке заявки. Страница бронирования тура представлена на рисунке 1.10.

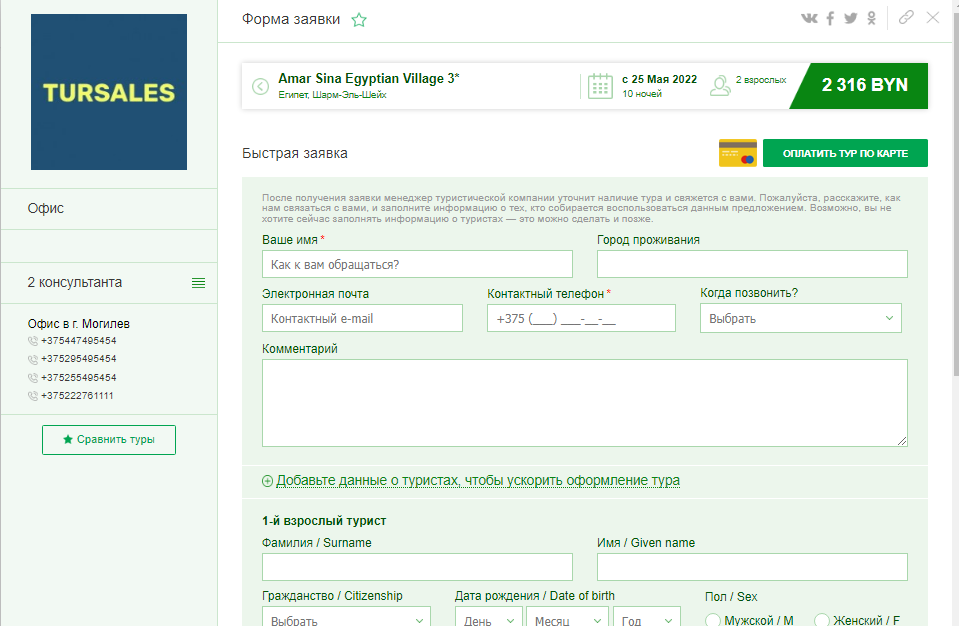


Рисунок 1.10 – Страница бронирования тура

Данное приложение в основном повторяет все возможности рассмотренного ранее *Tez-tour*, из недостатков стоит выделить:

* отсутствие возможности создать аккаунт;
* отсутствие поддержки, в которой можно задать вопрос;
* сложности при отслеживании заказа;
* дополнительные сборы после оформления заказа;
  + 1. **Туристическое агентство *Peopletravel.by***

Агентство *peopletravel.by* – это белорусская компания, которая предоставляет комплекс услуг для людей, предпочитающих в качестве досуга путешествия [5].

На главной странице сайта, можно выбрать тип тура для поиска:

* авиа/пакетные;
* экскурсионные;
* горящие.

Также можно подбирать туры по странам.

Главная страница сайта представлена на рисунке 1.11

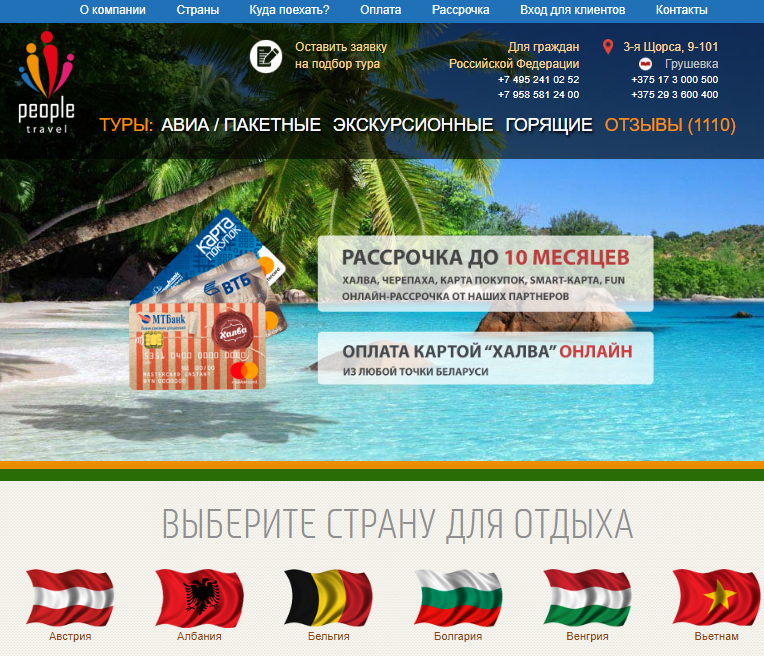


Рисунок 1.11 – Страница бронирования тура

Также туристическая компания *People Travel* предоставляет как бесплатные, так и платные услуги авиатранспортного предприятия. Основная из них – это авиаперевозки. Также предоставляются услуги по подбору максимально удобных авиаперелетов. При необходимости подбирается авиамаршрут туда-обратно. Еще предоставляется возможность заказать желаемый класс обслуживания. Страница с подбором авиатуров представлена на рисунке 1.12.



Рисунок 1.12 – Страница с подбором авиатуров

Можно выделить следующую особенность данного сайта, это возможность получить помощь в подборе туров от специалиста. Для этого необходимо оставить заявку на подбор тура, где можно выбрать тип отдыха, который интересует пользователя, и удобное время для звонка. Данная страница представлена на рисунке 1.13.

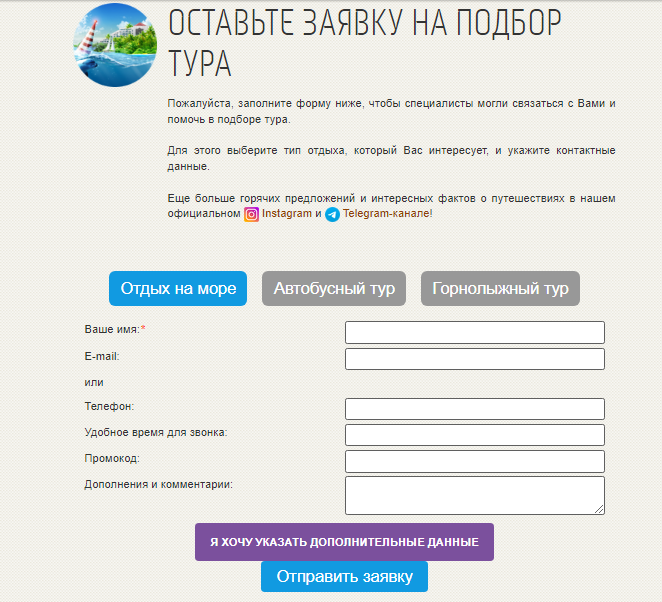


Рисунок 1.13 – Страница с подбором авиатуров

На сайте есть возможность оставить заявку на бронированию тура, для этого сначала необходимо выбрать тур, указать даты и количество человек. После этого появиться форма быстрой заявки, в которой необходимо указать контактные данные, а также добавить сведенья о туристах и паспортные данные, которые понадобятся при бронировании тура. Страница с формой заявки представлена на рисунке 1.14.



Рисунок 1.14 – Страница с формой заявки

Также стоит отменить, что данный сайт имеет хорошую репутацию среди своих пользователей. На сайте собрано более 1 110 отзывов со средней оценкой в 9.7 баллов.

Функционал сайта включает в себя:

* возможность выбора экскурсионного, авиа или круизного тура;
* наличие туристического календаря;
* возможность получения рассрочки;
* есть возможность сравнения туров;
* возможность найти ответы на часто задаваемые вопросы.
* возможность оставить заявку на подбор тура со специалистом;
* возможность задать вопрос в чате.

Однако данный сайт не лишён недостатков:

* нет возможности создать аккаунт;
* сложность в отслеживании забронированных туров;
* незначительные баги с отображением информации.

Таким образом, рассмотренные недостатки и достоинства аналогов должны быть учтены при разработке программного средства.

* 1. **Постановка задачи**

Основной задачей дипломного проекта является разработка веб-приложения «Туристическое агентство». Данное приложение должно предоставлять удобный интерфейс пользователю для выбора и бронирования туристических мест.

Основные задачи, которые необходимо реализовать в рамках дипломного проектирования:

* авторизация и регистрация пользователей;
* создания, удаления и редактирования туров администратором;
* добавление достопримечательностей к турам;
* создания, удаления и редактирования категорий туров администратором;
* отображение достопримечательностей на карте;
* добавление комментариев пользователями;
* одобрения и удаления комментариев пользователей администратором;
* расширенный поиск туров;
* оформление заказа и просмотр заказанных туров;
* возможность задать вопрос в чате;
* возможность редактирования профиля;
* просмотр статистики администратором.
  1. **Вывод по разделу**

В данном разделе был проведен аналитический обзор существующих веб-приложений туристических агентств. В качестве аналогов разрабатываемого приложения были рассмотрены следующие веб-приложения: *Tez-tour*, *Tury.by, Tursales.by* и *Peopletravel.by*. Выявлены их достоинства и недостатки.

Основываясь на рассмотренных примерах, было принято решение создания веб-приложения, предоставляющие удобный поиск и бронирование билетов для пользователей, а также широкий функционал для наполнения информации о турах администратором.

Необходимо реализовать серверную и клиентскую часть приложения.

Клиентская часть приложения должна: обладать удобным интерфейсом, предоставлять информацию о турах, быть легко поддерживаемой.

Серверная часть приложения должна: обеспечивать хранение данных, основываться на REST-архитектуре, обеспечивать легкую поддержку.

1. **Проектирование веб-приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 02.00.ПЗ

Разраб.

*Конопацкий А.А. АА.А.Конопацкий А. А.Конопацкий А. А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

2 Проектирования

веб-приложения

Лит.

Листов

12

*74417039, 2022*

* 1. **Используемые средства разработки**

Для разработки программного средства была выбрана интегрированная среда разработки *Visual Studio Code*, функционирование которой гарантирует необходимый уровень качества используемого кода на протяжении всего циклического процесса разработок программного обеспечения – от проектирования до непосредственной разработки

*Visual Studio Code* – [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0), разработанный [*Microsoft*](https://ru.wikipedia.org/wiki/Microsoft) для [*Windows*](https://ru.wikipedia.org/wiki/Windows), [*Linux*](https://ru.wikipedia.org/wiki/Linux) и *macOS*. Позиционируется как «лёгкий» редактор кода для [кроссплатформенной](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) разработки [веб](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5)- и [облачных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%87%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) приложений. Включает в себя [отладчик](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA), инструменты для работы с [*Git*](https://ru.wikipedia.org/wiki/Git)*,* [подсветку синтаксиса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%81%D0%B2%D0%B5%D1%82%D0%BA%D0%B0_%D1%81%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81%D0%B0), [*IntelliSense*](https://ru.wikipedia.org/wiki/IntelliSense) и средства для [рефакторинга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3).

Возможности *Visual Studio Code*:

* встроенные инструменты интеграции с *GitHub, GIT*, а также *Visual Studio Team Services* для быстрого тестирования, сборки, упаковки и развертывания разных типов приложений;
* работа с *Mono* и *Node.js* с помощью встроенного отладчика;
* поддержка *TypeScript* и *JavaScript*;
* публикация созданных приложений в *Microsoft Azure* через сервис *Visual Studio Team Services*;
* поддержка практически всех языков программирования;
* обширная библиотека шаблонов, готовых фрагментов кода и сниппетов с возможностью добавления своих элементов;
* одновременная работа с несколькими проектами (в нескольких окнах);
* интерфейс можно разделить на две панели для сравнения кода;
* функция отладки.
  1. **Архитектура веб-приложения**

Архитектура программного средства – это его строение как оно видно (или должно быть видно) извне его, т. е. представление программного средства как системы, состоящей из некоторой совокупности взаимодействующих подсистем. В качестве таких подсистем выступают обычно отдельные программы или сервисы, которые настроены и связаны определённым образом, в целях обеспечения необходимых характеристик, таких как производительность, отказоустойчивость, масштабируемость и безопасность. Разработка архитектуры является первым этапом борьбы со сложностью программного средства.

Архитектура программного средства представлена на рисунке 2.1.

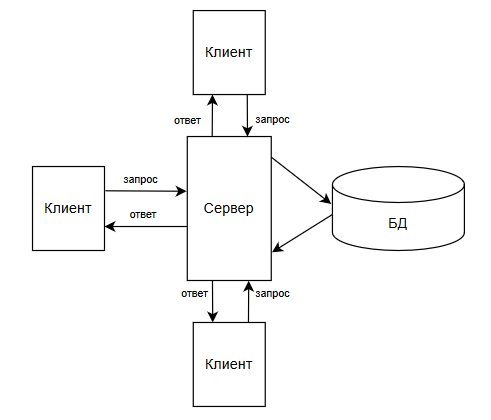


Рисунок 2.1 – Архитектура программного средства «Туристическое агентство»

Клиент предоставлен браузерным приложением, которое посылает запросы (*request*) по сети с использованием *HTTP* протокола, а сервер должен, принимать запросы от клиента и давать ответ (*response*). В нашем случае приложение разделено на две части. Это *frond-end* (клиентская) и *back-end* (серверная) части. Клиентская часть приложения имеет архитектуру *Single Page Application*, которую чаще всего упоминают как *SPA*.

В *SPA* приложении мы запрашиваем основную *HTML* страницу только один раз, а дальше мы общаемся с сервером с помощью *AJAX* технологии. Клиент видит, будто он общается в оконном приложении без перезагрузки страницы.

Принцип работы *SPA* приложения представлен на рисунке 2.2.

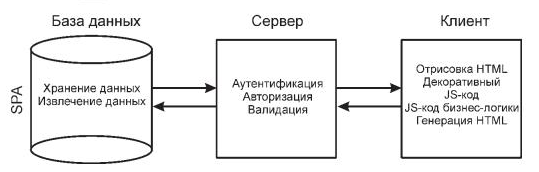
****

Рисунок 2.2 – Схема *SPA* приложения

Также для реализации будет использоваться паттерн *MVC*.

Паттерн *MVC* включает ряд компонентов:

* модели определяют структуру и логику используемых данных;
* представления (*views*) определяет визуальную часть, как данные будут отображаться;
* контроллеры обрабатывают входящие *http*-запросы, используя для обработки модели и представления, и отправляет в ответ клиенту некоторый результат обработки.

Система маршрутизация как дополнительный компонент сопоставляет запросы с маршрутами и выбирает для обработки запросов определенный контроллер.

В общем случае, когда к приложению приходит запрос, система маршрутизации выбирает нужный контроллер для обработки запроса. Контроллер обрабатывает запрос. В процессе обработки он может обращаться к данным через модели. Результат обработки контроллера отправляется в ответ клиенту.

На рисунке 2.3 показано взаимодействие между слоями в *MVC*. Шаблон *Model-View-Controller* для веб-приложений обеспечивает интеграцию с веб-сервером и сервером баз данных.

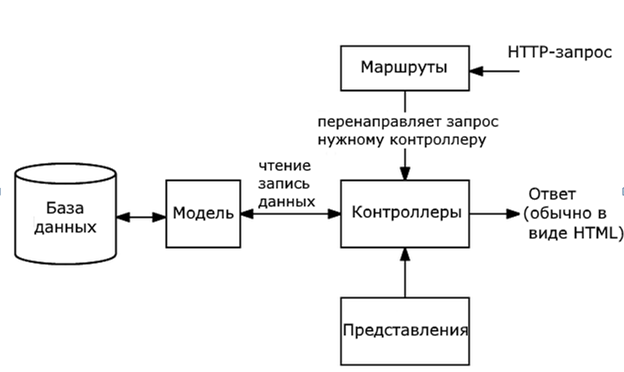


Рисунок 2.3 – Схема работы паттерна *MVC*

Разработанное приложение не является классическим представителем *MVC*. Проблема в слое *view*. Так как мы используем *SPA* подход в клиентской части, то роль *view* берет на себя *JSON* ответ вместо конкретной *HTML* страницы под конкретную акцию.

Схему архитектуры веб-приложения представлена в приложении А.

* 1. **Выбор средств реализации программных средств**

В качестве языка программирования был выбран язык *Javascript,* который отличается своей простотой, гибкостью в использовании [6]. На текущий момент *Javascript* динамично развивается, в открытом доступе есть большое количество библиотек и фреймворков. К преимуществам можно отнести интерпретируемость языка, не требуется каких-либо затрат на компиляцию, высокая скорость разработки кода, удобочитаемость и универсальность.

Для реализации серверной части была выбрана платформа *Node.js. Node* или *Node.js* – программная платформа, основанная на движке *V8*, превращающая *JavaScript* из узкоспециализированного языка в язык общего назначения [7]. *Node.js* добавляет возможность *JavaScript* взаимодействовать с устройствами [ввода-вывода](https://ru.wikipedia.org/wiki/%D0%92%D0%B2%D0%BE%D0%B4-%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4) через свой [*API*](https://ru.wikipedia.org/wiki/API), написанный на [*C++*](https://ru.wikipedia.org/wiki/C%2B%2B), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из *JavaScript*-кода. В основе *Node.js* лежит [событийно-ориентированное](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D0%B9%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [асинхронное (или реактивное)](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) программирование с [неблокирующим вводом/выводом](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B0%D1%8F_%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F).

Для работы также понадобится *NPM* (*node package manager*), который позволяет устанавливать необходимые библиотеки для реализации обработки запросов.

Для обработки запросов существует множество разный библиотек и фреймворков, таких *как Koa.js, Nest.js, Http, Meteor.js, Socket.io, Sails.js, Total.js*. В данном приложении будет использоваться фреймворк *Express.js* [8]. *Express* – это минималистичный и гибкий веб-фреймворк для приложений *Node.js*, предоставляющий обширный набор функций для мобильных и веб-приложений. Также использовалась библиотека *Socket.io* для работы чата [9]. *Socket.io* – *javascript*-библиотека для веб-приложений и обмена данными в реальном времени. Состоит из двух частей: клиентской, которая запускается в браузере и серверной для node.js. Оба компонента имеют похожее *API*.

Для разработки клиентской части был также выбран язык *Javascript* и библиотека *React.js* [10]. ***React.js*** это библиотека ***Javascript*** с открытым исходным кодом для построения компонентов интерфейса, которые можно переиспользовать. *React* разрабатывается и поддерживается [*Facebook*](https://ru.wikipedia.org/wiki/Facebook), [*Instagram*](https://ru.wikipedia.org/wiki/Instagram) и сообществом отдельных разработчиков и корпораций. *React* может использоваться для разработки [одностраничных](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) и мобильных приложений. Его цель – предоставить высокую скорость, простоту и [масштабируемость](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C). В качестве библиотеки для разработки пользовательских интерфейсов *React* часто используется с другими библиотеками, такими как *MobX,*[*Redux*](https://ru.wikipedia.org/wiki/Redux)*и*[*GraphQ*](https://ru.wikipedia.org/w/index.php?title=GraphQL&action=edit&redlink=1).

Особенности *React*.*js*:

* однонаправленная передача данных
* виртуальный *DOM*;
* использование *JSX* (*JavaScript* *XML*);
* методы жизненного цикла;
* *react* *hooks*;
* универсальность.

Однонаправленная передача данных позволяет передавать свойства от родительских элементов к дочерним. Компоненты получают свойства как множество [неизменяемых](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *immutable*) значений, поэтому компонент не может напрямую изменять свойства, но может вызывать изменения через [*callback*-функции](https://ru.wikipedia.org/wiki/Callback_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Такой механизм называют «свойства вниз, события наверх».

Виртуальный *DOM* создаёт [кэш](https://ru.wikipedia.org/wiki/%D0%9A%D1%8D%D1%88)-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления *DOM* браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить.

*JSX* позволяет расширить синтаксис *JavaScript*, который даёт возможность использовать *HTML*-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием *JSX*, но также есть возможность использования обычного [*JavaScript*](https://ru.wikipedia.org/wiki/JavaScript).

Методы жизненного цикла позволяют разработчику запускать код на разных стадиях жизненного цикла компонента. Например:

* *shouldComponentUpdate* – позволяет предотвратить перерисовку компонента с помощью возврата *false*, если перерисовка не нужна;
* *compoenentDidMount* – вызывается после первой отрисовки компонента. Часто используется для запуска получения данных с удалённого источника через [*API*](https://ru.wikipedia.org/wiki/API);
* *render* – важнейший метод жизненного цикла. Каждый компонент должен иметь этот метод. Обычно вызывается при изменении данных компонента для перерисовки данных в интерфейсе.

Для хранения состояний используется хранилище *Redux* [11].

*Redux* – библиотека для *JavaScript* с открытым исходным кодом, предназначенная для управления состоянием приложения. Чаще всего используется в связке с *React* или *Angular* для разработки клиентской части. Содержит ряд инструментов, позволяющих значительно упростить передачу данных хранилища через контекст. *Redux* упрощает хранение данных и передачу их в компоненты *React*.

* 1. **Диаграмма вариантов использования**

Для начала разработки необходимо спроектировать диаграмму вариантов использования. Диаграмма вариантов использования описывает динамические аспекты поведения системы в виде блок-схемы и отражает бизнес-процессы, логику процедур и потоки работ – переходы от одной деятельности к другой.

В данном приложении гость взаимодействует с четырьмя прецедентами: вход в систему, создание аккаунта, поиск туров и добавление тура в корзину.

Далее пользователь – это актёр, который выполнил вход в систему. У него гораздо больше возможностей по отношению к гостю. Из них можно выделить просмотр личного профиля, где пользователь может изменить настройки профиля, возможность написать в чат поддержки, просмотр корзины, возможность забронировать тур, с последующей возможностью написать комментарий.

Далее администратор. Это актёр, который может добавлять, обновлять и удалять туры, достопримечательности к турам, категории туров, а также страны и города. Дополнительно к этому администратор может просматривать статистику сайта, отвечать пользователям в чате на их вопросы, а также одобрять и удалять комментарии.

Благодаря данной диаграмме в процессе разработки будет легче понимать, что из себя должно представлять веб-приложение.

Диаграмма вариантов использования представлена в приложении Б.

* 1. **Проектирование базы данных**

*MongoDB* – система управления базами данных, которая работает с документоориентированной моделью данных. В отличие от реляционных СУБД, в *MongoDB* не требуются таблицы, схемы или отдельный язык запросов [12]. Информация хранится в виде документов либо коллекций.

Разработчики позиционируют продукт как промежуточное звено между классическими СУБД и *NoSQL*. *MongoDB* не использует схемы, как это делают реляционные базы данных.

Особенности *MongoDB*:

* однонаправленная передача данных;
* кроссплатформенность. СУБД разработана на языке программирования С++, поэтому с легкостью интегрируется под любую операционную систему;
* формат данных. *MongoDB* использует собственный формат хранения информации – *Binary JavaScript Object Notation* (*BSON*), который построен на основе языка *JavaScript*;
* документ. Если реляционные БД используют строки, то *MongoDB* – документы, которые хранят значения и ключи;
* вместо таблиц *MongoDB* использует коллекции. Они содержат разные типы наборов данных;
* репликация. Система хранения информации в СУБД представлена узлами. Существует один главный и множество вторичных. Данные реплицируются между точками. Если один первичный узел выходит из строя, то вторичный становится главным;
* индексация. Технология применяется к любому полю в документе на усмотрение пользователя. Проиндексированная информация обрабатывается быстрее;
* для сохранения данных большого размера *MongoDB* использует собственную технологию *GridFS*, состоящую из двух коллекций. В первой (*files*) содержатся имена файлов и метаданные по ним. Вторая (*chunks*) сохраняет сегменты информации, размер которых не превышает 256 Кб;
* СУБД осуществляет поиск по специальным запросам. Например, пользователь может создать диапазонный запрос и мгновенно получить ответ.
* балансировщик нагрузки используется в СУБД не только для распределения нагрузки между разными базами данных, но и для горизонтального масштабирования. Сегменты БД распределяются по разным узлам, что повышает производительность. При этом базы данных, расположенные на разных узлах, синхронизированы между собой и обеспечивают целостность информации для клиента;
* *mongoDB* может поставляться для конечного клиента как облачное решение.

СУБД используют для хранения событий в системе (логирование), записи информации с датчиков мониторинга на предприятии, а также в сфере электронной коммерции и мобильных приложений.

Для начала разработки проекта необходимо спроектировать базу данных. Чтобы составить визуальную взаимосвязанную структуру базы данных, нам необходимо продумать, какая информация будет храниться в этих коллекциях, после этого определить структуру коллекций.

Диаграмма базы данных представляет собой коллекции, которые содержат названия и тип полей.

В ходе проектирования была создана схема базы данных, которая продемонстрирована в приложении В.

База данных приложения состоит из 15 коллекций. Поля в коллекциях могут включать в себя другие коллекции, массивы, объекты, либо ссылки на другие коллекции.

Коллекция *Tours* хранит информацию о турах. Описание коллекции представлено в таблице 2.1.

Таблица 2.1 – Коллекция *tours*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *id* | *Id* | Идентификатор коллекции |
| *Title* | *String* | Название тура |
| *Category* | *Id* | Категория тура |
| *Label* | *String* | Короткое название |
| *Desc* | *String* | Короткое описание |
| *additionalInfo* | *String* | Информация о туре |
| *Price* | *Number* | Цена |
| *Rating* | *Number* | Рейтинг |
| *numReviews* | *Number* | Количество отзывов |
| *Country* | *countryId* | Страна тура |
| *City* | *cityId* | Город тура |
| *Comments* | *[commentsSchema]* | Комментарии |
| *Attraction* | *[attractionSchema]* | Достопримечательности |
| *Category* | *categoryId* | Категория |
| *imageGallery* | *[String]* | Массив изображений |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *Countries* хранит информацию о странах. Описание коллекции представлено в таблице 2.2.

Таблица 2.2 – Коллекция *Countries*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *countryName* | *String* | Название страны |
| *countryDesc* | *String* | Описание страны |
| *countryImage* | *String* | Изображение страны |

Коллекция *Cities* хранит информацию о странах. Описание коллекции представлено в таблице 2.3.

Таблица 2.3 – Коллекция *Cities*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *cityName* | *String* | Название города |
| *cityDesc* | *String* | Описание города |
| *cityImage* | *String* | Изображение города |
| *Country* | *Id* | Название страны |
| *Lon* | *Number* | Долгота |
| *Lat* | *number* | Широта |

Коллекция *Categories* хранит информацию о категориях туров. Описание коллекции представлено в таблице 2.4.

Таблица 2.4 – Коллекция *Categories*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *categoryName* | *String* | Название категории |
| *categoryDesc* | *String* | Описание категории |
| *categoryImage* | *String* | Изображение |

Коллекция *Comments* хранит информацию о комментариях. Описание коллекции представлено в таблице 2.5.

Таблица 2.5 – Коллекция *Comments*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *id* | *Id* | Идентификатор коллекции |
| *User* | *userId* | Идентификатор пользователя |
| *Comment* | *String* | Комментарий |
| *Rating* | *Number* | Оценка от пользователя |
| *isActive* | *Boolean* | Статус комментария |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *Attraction* хранит информацию о достопримечательностях. Описание коллекции представлено в таблице 2.6.

Таблица 2.6 – Коллекция *Attraction*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *id* | *Id* | Идентификатор документа |
| *titleAttraction* | *String* | Название достопримечательности |

Окончание таблицы 2.6

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *descAttraction* | *String* | Описание |
| *imageAttraction* | *String* | Изображение |
| *adressAttraction* | *String* | Адрес места |
| *Lon* | *Number* | Долгота |
| *Lat* | *Number* | Широта |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *AvailableSeats* хранит даты и доступные места. Описание коллекции представлено в таблице 2.7.

Таблица 2.7 – Коллекция *AvailableSeats*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *startDate* | *Date* | Дата начала |
| *endDate* | *Date* | Дата окончания |
| *availableSeats* | *Number* | Общее количество мест |
| *reservedSeats* | *Number* | Кол-во зарезервированных мест |

Коллекция *Users* хранит информацию о пользователях. Описание коллекции представлено в таблице 2.8.

Таблица 2.8 – Коллекция *Users*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *id* | *Id* | Идентификатор документа |
| *firstName* | *String* | Имя пользователя |
| *lastName* | *String* | Фамилия пользователя |
| *Email* | *String* | Email адрес пользователя |
| *Password* | *String* | Пароль пользователя |
| *isAdmin* | *Boolean* | Флаг, который указывает, является ли пользователь администратором |
| *isActive* | *Boolean* | Флаг, который указывает статус пользователя |
| *imageProfile* | *String* | Изображение пользователя |
| *verificationInfo* | *verificationSchema* | Схема верификации |
| *passwordResetInfo* | *passwordResetSchema* | Схема сброса пароля |
| *Verified* | *Boolean* | Статус верификации |
| *Oauth* | *String* | Тип авторизации |
| *visitShema* | *visitShema* | Схема посещений туров |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *VerificatioInfo* хранит информацию о верификации пользователя. Описание коллекции представлено в таблице 2.9.

Таблица 2.9 – Коллекция *VerificatioInfo*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *uniqueString* | *String* | Уникальная строка |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *PasswordReset* хранит информацию о cбросе пароле пользователя. Описание коллекции представлено в таблице 2.10.

Таблица 2.10 – Коллекция *PasswordReset*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *uniqueString* | *String* | Уникальная строка |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время обновления |

Коллекция *VisitSchema* хранит информацию о посещениях туров. Описание коллекции представлено в таблице 2.11.

Таблица 2.11 – Коллекция *VisitSchema*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *tourId* | *Id* | Идентификатор тура |
| *Counter* | *Number* | Счётчик посещений |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *Orders* хранит информацию о заказах. Описание коллекции представлено в таблице 2.12.

Таблица 2.12 – Коллекция *Orders*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *ordersInfo* | *[ordersInfo]* | Массив заказов |
| *totalPrice* | *Number* | Общая стоимость |
| *userInfo* | *userId* | Идентификатор пользователя |
| *isPaid* | *Boolean* | Статус оплаты |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *OrdersInfo* хранит дополнительную информацию о заказах. Описание коллекции представлено в таблице 2.13.

Таблица 2.13 – Коллекция *OrdersInfo*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *tourId* | *Id* | Идентификатор тура |
| *startDate* | *Date* | Дата начала тура |
| *endDate* | *Date* | Дата окончания тура |
| *numberOfPerson* | *Number* | Количество человек |
| *totalPrice* | *number* | Общая стоимость |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Коллекция *Chat* хранит информацию о чатах пользователей. Описание коллекции представлено в таблице 2.14.

Таблица 2.14 – Коллекция *Chat*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *Id* | *Id* | Идентификатор документа |
| *userId* | *Id* | Идентификатор пользователя |
| *Name* | *String* | Имя пользователя |
| *lastName* | *String* | Фамилия пользователя |
| *isAdmin* | *Boolean* | Статус администратора |
| *Online* | *Boolean* | Статус пользователя |
| *socketId* | *String* | Идентификатор сокета |
| *Messages* | *[messagesShema]* | Массив сообщений |

Коллекция *messageSchema* хранит информацию о сообщениях. Описание коллекции представлено в таблице 2.15.

Таблица 2.15 – Коллекция *messageSchema*

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| *userId* | *Id* | Идентификатор пользователя |
| *Body* | *String* | Сообщение |
| *Name* | *String* | Имя пользователя |
| *lastName* | *String* | Фамилия пользователя |
| *isAdmin* | *boolean* | Статус администратора |
| *online* | *Boolean* | Статус пользователя |
| *socketId* | *String* | Идентификатор сокета |
| *createdAt* | *Date* | Дата и время создания |
| *updatedAt* | *Date* | Дата и время последнего обновления |

Для отображения данных с БД в объектное представление была использована *ODM* *Mongoose* [13].

* 1. **Выводы по разделу**

Для разработки веб-приложения был выбран язык программирования Javascript, платформа *Node*.*js*, так как они обеспечивают наибольший выигрыш в скорости и гибкости разработки, обладают развитой базой готовых решений для типичных и ряда специфических задач, а также хорошей документированностью и развитым сообществом, которое позволяет на порядок быстрее искать решения на возникающие в процессе изучения, в том числе глубокого, платформы и разработки приложений вопросы и трудности.

Для клиентской части веб-сервиса был выбран язык *Javascript* и библиотеки *React*.*js* и *Redux*, так как они имеют гибкую архитектуру и позволяют в короткие сроки разработать веб-приложение.

В качестве системы управления базой данных выбор был сделан в пользу *MongoDB*.

Также в данном разделе была спроектирована структура базы данных.

Выбранные технологии позволят разработать гибкую систему способную продуктивно работать при больших нагрузках и не потребуют больших затрат ресурсов сервера.

1. **Разработка веб-приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 03.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

3 Разработка

веб-приложения

Лит.

Листов

11

*74417039, 2022*

Разрабатываемое приложение включает в себя две части: серверную и клиентскую. Для разработки каждой из них будет использован свой набор технологий, включающий различные библиотеки, языки программирования.

* 1. **Разработка серверной части**

Главная цель разрабатываемого приложения – это поиск туров и бронирование заказа, а также возможность поиска определенных туров по нужным параметрам. В ходе разработки необходимо создать базу данных, которая удовлетворит поставленным задачам. Данная реализация должна быть достаточно проста и универсальна, чтобы с ее помощью в дальнейшем можно было изменять и дополнять структуру базы данных. Функционально программное средство должно выполнять следующие задачи:

* аутентификация пользователей;
* добавление туров и наполнение их информацией;
* просмотр и поиск туров;
* бронирование туров;
* возможность задать вопрос в чате;
* просмотр статистики приложения.

Общая структура проекта представлена на рисунке 3.1.

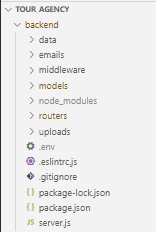
****

Рисунок 3.1 – Структура серверной части

Корневая директория содержит следующие поддиректории:

* *emails* содержит настроенный модуль *nodemailer*, для отправки сообщений по почте;
* *middleware* включает в себя функция промежуточной обработки;
* *models* описывает модели базы данных;
* *node\_modules* хранит необходимые библиотеки для работы приложения;
* *routers* содержит в себе файлы, обрабатывающие входящие *http*-запросы и вызывающие определённые сервисы;
* *uploads* каталог, где хранятся изображения загруженные на сервер.

Директория содержит следующие файлы:

* *package*.*json* и *package*-*lock*.*json* содержат все важные данные о структуре зависимостей проекта и их версий;
* *env* хранит переменные среды приложения;
* .*eslintrc*.*js* содержит настройки для библиотеки *eslint*, которая помогает автоматически форматировать код;
* *gitignore* необходим для системы контроля версий *Git*;
* *server*.*js* основной файл, который необходим для запуска сервера.

Во время разработки серверной части использовался архитектурный стиль *REST*. Одним из принципов *REST* является независимость от состояния. Это значит, что клиент должен сам позаботиться о аутентификации при каждом запросе. Для выполнения этой задачи был применён *JSON* *Web* *Token* (*JWT*).

*JSON* *Web* *Token* – это открытый стандарт для создания токенов доступа, основанный на формате [*JSON*](https://ru.wikipedia.org/wiki/JSON). Как правило, используется для передачи данных для [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) в клиент-серверных приложениях. JWT состоит из трех частей: заголовок *header*, полезные данные *payload* и подпись *signature*. Давайте пройдемся по каждой из них. Хедер JWT содержит информацию о том, как должна вычисляться JWT подпись. ***Payload*** – это полезные данные, которые хранятся внутри JWT. *Signature* вычисляется на основании первых и зависит от выбранного алгоритма (в случае использования неподписанного *JWT* может быть опущен).

Реализацию *JWT* токена в приложении можно увидеть на листинге 3.1.

|  |
| --- |
| const jwt = require('jsonwebtoken')  module.exports = {  generateJsonToken: function(user) {  return jwt.sign(  {\_id: user.\_id,  firstName: user.firstName,  lastName: user.lastName,  email: user.email,  isAdmin: user.isAdmin},  process.env.JWT\_TOKEN || '73UCuYCi',{  expiresIn: '30d'  } ) } } |

Листинг 3.1 – Реализация генерации *JWT* токена

*JWT* токен будет передаваться в каждом запросе, где требуется проверка доступа.

Описание модели *Order* представлено на листинге 3.2.

|  |
| --- |
| const { Schema, model, mongoose } = require('mongoose')  const ordersInfo = new Schema(  {  \_id: { type: Schema.Types.ObjectId, ref: 'Tours', required: true },  startDate: { type: Date, required: true },  endDate: { type: Date, required: true },  numberOfPerson: { type: Number, required: true },  numberOfDays: { type: Number, required: true },  totalPrice: { type: Number, required: true },  },  {timestamps: true,}  )  const orderSchema = new Schema({  ordersInfo: [ordersInfo],  paymentMethod: { type: String, required: false },  totalPrice: { type: Number, required: false },  userInfo: { type: Schema.Types.ObjectId, ref: 'Users', required: true },  isPaid: { type: Boolean, default: false },  paidAt: { type: Date },  },  {timestamps: true,}  )  module.exports = model('Orders', orderSchema) |

Листинг 3.2 – Описание модели *Order*

Реализация функции роутера для бронирования тура представлена на листинге 3.3.

|  |
| --- |
| orderRouter.post('/', isAuth, async (req, res) => {  if (req.body.orderItems.length === 0) {  res.status(400).send({ message: 'Cart is empty' })  } else {  let totalPrice = 0  req.body.orderItems.forEach(el =>{totalPrice += el.totalPrice})  const order = new Order({  ordersInfo: req.body.orderItems,  userInfo: req.user.\_id,  totalPrice})  const createdOrder = await order.save()  res.status(201).send({ message: 'Order created', order: createdOrder })}}) |

Листинг 3.3 – Реализация роутера бронирования тура

Для того, чтобы забронировать тур, пользователь должен сначала выбрать тур, который идентифицируется по уникальному полю *id*, затем выбрать диапазон дат и количество человек. При бронировании тура указывается стоимость за один день, таким образом, чтобы рассчитать полную стоимость, необходимо перемножить стоимость на количество дней и количество человек.

Реализация роутера для редактирования тура представлена на листинге 3.4.

|  |
| --- |
| tourRouter.put('/:id', isAuth, isAdmin, async (req, res) => {  const categoryFrom = await Category.findOne({categoryName: req.body.category})  const tourId = req.params.id  const tour = await Tour.findById(tourId)  if (tour) {  tour.title = req.body.title;  tour.image = req.body.image;  tour.category = req.body.category;  tour.label = req.body.label;  tour.desc = req.body.desc;  tour.additionalInfo = req.body.additionalInfo;  tour.price = req.body.price;  tour.country = req.body.country;  tour.city = req.body.city;  tour.imageGallery = req.body.uploadedImage.length !== 0 ? req.body.uploadedImage : tour.imageGallery,  tour.categoryS = categoryFrom.\_id  const updatedTour = await tour.save()  res.send(  { message: 'Tour updated', tour: updatedTour}  )  } else {  res.status(404).send(  { message: 'Tour not found' }  )  }  }  ) |

Листинг 3.4 – Роутер для редактирования тура

Для добавления комментарий пользователь должен быть авторизован, для проверки авторизации используется промежуточный обработчик *isAuth,* затем необходимо проверить, чтобы в заказах пользователя был выбранный тур, после этого можно переходить к созданию объекта *comment* и к обновлению тура. В случае успешного добавления, пользователь получит ответ со статусом 201. Если же в заказах у пользователя не было этого тура, то он получит ответ со статусом 404.

Реализация функции добавления комментарий представлена на листинге 3.5.

|  |
| --- |
| tourRouter.post('/:id/comments', isAuth, async (req, res) => {  const tourId = req.params.id  const tour = await Tour.findById(tourId)  if (tour) {  const checkOrder = await Order.find({ userInfo : req.user.\_id, orderItems: tourId } )  if (checkOrder.length !== 0 || req.user.isAdmin) {  const comment = {user: req.user.\_id,  comment: req.body.comment,  rating: Number(req.body.rating) }  tour.comments.push(comment)  tour.numReviews = tour.comments.length  tour.rating = (tour.comments.reduce((a, c) => c.rating + a, 0 ) / tour.comments.length).toFixed(2)  const updatedTour = await tour.save()  res.status(201).send({ message: 'Комментарий добавлен', comment: updatedTour.comments[updatedTour.comments.length - 1]})  } else {res.status(404).send({ message: 'Вы не можете писать комментарий, так как вы не заказывали данный тур.' })}  } else { res.status(404).send({ message: 'Tour not found' }) } }) |

Листинг 3.5 – Реализация функции добавления комментария

Реализация обработки события получения сообщения на сервере представлена на листинге 3.6.

|  |
| --- |
| socket.on('onMessage', async message => {  if (message.isAdmin) {  const user = await Chat.findOne({ \_id : message.\_id, online : true })  if (user) {  io.to(user.socketId).emit('message', message)  user.messages.push(message)  await user.save()}  } else {  const admin = await Chat.findOne({ isAdmin : true, online: true} )  if (admin) {  io.to(admin.socketId).emit('message', message)  const user = await Chat.findOne({ \_id : message.\_id, online : true})  user.messages.push(message)  await user.save()  } else {  io.to(socket.id).emit('message', {  name: 'Admin',  body: 'Sorry. I am not online right now'}  )} }  }) |

Листинг 3.6 – Обработка события получения сообщения

Блок-схему алгоритма бронирования туров можно увидеть в приложении Г.

Диграмма классов серверной части веб-приложения представлена в приложении Д.

Листинг роутра туров представлен в приложении И.

Листинг роутера панели администратора представлен в приложении К.

* 1. **Разработка клиентской части**

В рамках создания клиентской части веб-приложения туристическое агентство, был создан веб-интерфейс, который позволяет использовать весь необходимый функционал из разработанного серверного приложения. На рисунке 3.2 предоставлена общая структура проекта клиентской части разрабатываемого веб-приложения.

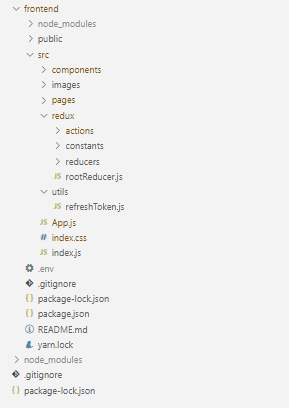
****

Рисунок 3.2 – Структура клиентской части

Разработка была начата с разработки основной структуры проекта клиентской части веб-приложения.

Корневая директория содержит следующие поддиректории и файлы:

* *src* – основная директория всего исходного кода клиентской части разработанного веб-приложения;
* *public* – папка, в которой находятся все загруженные пользователями фотографии;
* .*gitignore* – необходим для системы контроля версий *Git*;
* *package*.*json* – это файл, который хранит названия пакетов, их версии и зависимости;
* *package*-*lock*.*json* – это файл, который предназначен для блокировки зависимостей от определённого номера версии;.
* *env* хранит переменные среды приложения;
* .*eslintrc* – набор правил, описывающих нужную структуру написания кода и компонентов в проекте;
* *сomponents* – хранит файлы с React-компонентами;
* *pages* – хранит файлы с React-компонентами, выступающими в качестве страниц;
* *redux* – содержит в себе директории и файлы, необходимые для работы с *Redux*.

Для отображения рейтинга тура был разработан компонент *Rating*, который представлен на листинге 3.7.

|  |
| --- |
| import React from "react";  import \* as BsIcons from 'react-icons/bs'  export default function Rating(props) {  const { rating } = props  const drawStars = () => {  let content = [];  for (let i = 1; i <= 5; i++) {  content.push(  <span>  {  rating >= i  ?  <BsIcons.BsStarFill />  : rating >= i - 0.5  ? <BsIcons.BsStarHalf />  : <BsIcons.BsStar />  }  </span>  );  }  return content;  };  return (  <div className="rating star-rating">  {drawStars()}  <span></span>  </div>  )} |

Листинг 3.7 – Компонент *Rating*

Для использования компонента *Rating н*обходимо вызвать его и передать два параметра: *rating* (количество звёзд) и *numReviews* (количество комментарий). Пример использования компонента Rating представлен на листинге 3.8.

|  |
| --- |
| <Rating rating={tour.rating} numReviews={tour.numReviews} /> |

Листинг 3.8 – Применение компонента *Rating*

Для обработки начального состояния в *redux* используется специальный объект *initialState*. Пример представлен на листинге 3.9.

|  |
| --- |
| const initialState = {  userSignIn: {  userInfo: localStorage.getItem("userInfo")  ? JSON.parse(localStorage.getItem("userInfo"))  : null  },  cart: {  cartItems: localStorage.getItem('cartItems')  ? JSON.parse(localStorage.getItem('cartItems'))  : []  },  loginData: localStorage.getItem('loginData')  ? JSON.parse(localStorage.getItem('loginData'))  : null  }  const store = createStore(rootReducer, initialState, compose(  applyMiddleware(thunk),  window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_()  )  ) |

Листинг 3.9 – Обработка начального состояния в *redux*

Объект *initialState* представляет собой начальное состояние хранилища. Он является вторым не обязательным аргументом метода *createStore*(). С созданием хранилища можно сразу объявить начальное состояние для его полей. Этот объект желательно создавать, даже в тех случаях, когда объявления начального состояния не требуется. Потому что этот объект помогает посмотреть на структуру хранилища и название его полей. В представленном выше листинге, будет проверяться локальное хранилище браузера на наличие объектов с информацией о пользователе и его корзине. Если такие объекты есть, то они будет записаны в начальное состояние.

Для отображения компонентов, доступных только администратору, был реализован компонент *adminRoute*, представлен на листинге 3.10.

|  |
| --- |
| import React from 'react';  import { useSelector } from 'react-redux';  import { Redirect, Route } from 'react-router-dom';  import MessageBox from "../components/MessageBox";  export default function AdminRoute({ component: Component, ...rest }) {  const userSignIn = useSelector(state => state.userSignIn);  const { userInfo } = userSignIn;  return (  <Route  {...rest}  render={(props) =>  userInfo && userInfo.isAdmin ?  (<Component {...props}></Component> ) :  ( <div className='error403'><MessageBox variant="danger">Ошибка 403. В доступе отказано.</MessageBox></div> )  }  ></Route>  );  } |

Листинг 3.10 – Компонент *adminRoute*

Для отображения компонентов, доступных авторизованным пользователям, был реализован компонент *privateRoute*, представлен на листинге 3.11.

|  |
| --- |
| import React from 'react';  import { useSelector } from 'react-redux';  import { Redirect, Route } from 'react-router-dom';  export default function PrivateRoute({ component: Component, ...rest }) {  const userSignIn = useSelector((state) => state.userSignIn);  const { userInfo } = userSignIn;  return (  <Route  {...rest}  render={(props) =>  userInfo ?  ( <Component {...props}></Component> ) :  ( <Redirect to="/login" /> )  }  ></Route>  );} |

Листинг 3.11 – Компонент *privateRoute*

Для отображения карты и достопримечательностей на ней использовался компонент *Ymaps*, представлен на листинге 3.12.

|  |
| --- |
| <YMaps query={{ apikey: process.env.REACT\_APP\_API\_KEY\_YANDEX\_MAPS}}>  <Map  width='100%'  height='500px'  state={centerMap && centerMap}>  <Clusterer  options={{  preset: 'islands#invertedVioletClusterIcons',  groupByCoordinates: false,  clusterDisableClickZoom: true,  clusterHideIconOnBalloonOpen: false,  geoObjectHideIconOnBalloonOpen: false,}} >  {  tour.attractions.map((a, idx) =>  <Placemark  geometry={[a.lat, a.lon]}  key={idx}  options={getPointOptions()}  properties={getPointData(idx, a.titleAttraction, a.descAttraction, a.imageAttraction)}  modules={['geoObject.addon.balloon', 'geoObject.addon.hint']}/>)  }  </Clusterer>  <FullscreenControl />  <GeolocationControl options={{ float: 'left' }} />  <TypeSelector options={{ float: 'right' }} />  <ZoomControl options={{ float: 'right' }} />  </Map></YMaps> |

Листинг 3.12 – Компонент Ymaps

Использованный компонент *Ymaps* находится в сторонней библиотеке *react*-*yandex*-*map*. Для подключения полного функционала, было необходимо подключить *api-ключ* из кабинета разработчика в Яндекс.

При формировании интерактивной карты происходит динамическое формирование *HTML-*кода и внедрение его в *DOM*-дерево страницы, на которой размещена карта. *API* Яндекс.Карт разработан с использованием концепции модульности. Это позволяет настраивать *API* под конкретные задачи и предоставляет возможность регулировать объем загружаемого трафика.

При создании карты необходимо обязательно задать начальную область путем указания центра и уровня масштабирования.

Для отображения меток использовался вспомогательный класс *Placemark,* в который можно передать координаты места, стиль метки и содержимое контента, которое будет появляться при клике на метку.

Для обновления *refreshToken* при авторизации через почту *gmail* была разработана функция *refreshTokenSetup,* которая запускается асинхронно и обновляет токен по истечению времени его жизни. Данная функция представлена на листинге 3.13.

|  |
| --- |
| export const refreshTokenSetup = (res) => {  let refreshTiming = (res.tokenObj.expires\_in || 3600 - 5 \* 60) \* 1000;  const refreshToken = async () => {  const newAuthRes = await res.reloadAuthResponse();  refreshTiming = (newAuthRes.expires\_in || 3600 - 5 \* 60) \* 1000;  let userInfo = JSON.parse(localStorage.getItem('userInfo'));  userInfo.token = newAuthRes.access\_token  localStorage.setItem('userInfo', JSON.stringify(userInfo));  setTimeout(refreshToken, refreshTiming);  };    setTimeout(refreshToken, refreshTiming);  }; |

Листинг 3.13 – Функция *refreshTokenSetup*

Диаграмма последовательности работы чата представлена в приложении Е.

Скриншот работы разработанного приложения можно найти в приложении Ж.

Листинг *React*-компонента *Tour* представлен в приложении Л

* 1. **Выводы по разделу**

В данном разделе были подробно рассмотрены этапы разработки серверной и клиентской частей веб-сервиса.

Разработан веб-сервис, предназначенный для поиска и бронирования туров. В процессе разработки были подробно рассмотрены реализация контроллеров и сервисов серверной части, а также реализация компонентов представлений и компонентов-контейнеров клиентской частей веб-сервиса.

1. **Тестирование веб-приложения**

ФИО

Подпись

Дата

Лист

1

БГТУ 04.00.ПЗ

Разраб.

*Конопацкий А. А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

4 Тестирование

веб-приложения

Лит.

Листов

6

*74417039, 2022*

Тестирование программного обеспечения – это процесс исследования ПО с целью выявления ошибок и определения соответствия между реальным и ожидаемым поведением ПО, осуществляемый на основе набора тестов, выбранных определённым образом.

В более широком смысле, тестирование ПО – это техника контроля качества программного продукта, включающая в себя проектирование тестов, выполнение тестирования и анализ полученных результатов. Контроль качества (тестирование) считается важным в процессе разработки ПО, потому что обеспечивает безопасность, надёжность, удобство создаваемого продукта.

Надежность – способность программного обеспечения выполнять требуемые задачи в обозначенных условиях на протяжении заданного промежутка времени или указанное количество операций.

* 1. **Тестирование основного функционала веб-приложения**

Тестирование программного средства проводилось на основании функциональных требований, к программному продукту, представленных в первом разделе. При этом проверялось соответствие функциональности реализованных страниц и описаний требований. Было произведено тестирование основного функционала приложения. Помимо теста соответствия функционала описанию, было произведено тестирование приложения на наличие уязвимостей. Также было проведено негативное тестирование форм, по типу ввода неправильных данных, ручное тестирование и переход напрямую по ссылкам, не предназначенных для текущих типов пользователей. Некоторые из тестов основного функционала представлены в таблице 4.1.

Таблица 2.14 – Результаты тестирования основного функционала

|  |  |  |  |
| --- | --- | --- | --- |
| Название теста | Ожидаемый результат | Полученный результат | Результат теста |
| Регистрация пользователя | Пользователь создан, данные занесены в БД | Пользователь создан, данные занесены в БД | Прошёл |
| Авторизация пользователя | Пользователь получает доступ к функционалу приложению | Пользователь получил доступ к функционалу приложения | Прошёл |

Окончание таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название теста | | Ожидаемый результат | Полученный результат | Результат теста |
| Восстановление пароля | | Пользователь получает сообщение на почту для смены пароля | Пользователь получил сообщения для смены пароля | Прошёл |
| Управление турами | Туры создаются, изменяются, удаляются | | Туры создаются, изменяются, удаляются | Прошёл |
| Управление категориями туров | Категории создаются, изменяются, удаляются | | Категории создаются, изменяются, удаляются | Прошёл |
| Просмотр главной страницы с турами | На странице появится каталог популярных туров по странам | | На странице появились популярные туры по странам | Прошёл |
| Поиск туров через форму на главной странице | На странице появиться новый список туров | | На странице появился новый список туров | Прошёл |
| Просмотр тура | На странице появится информация с описанием тура, его стоимостью, достопримечательностями и галереей | | На странице появилась вся необходимая информация | Прошёл |
| Бронирование тура | Тур появиться в списке заказов | | Тур появился в списке заказов | Прошёл |
| Комментария к туру | Появится комментарий с пометкой о модерации | | Комментарий появился с пометкой о модерации | Прошёл |
| Расширенный поиск туров | Появятся туры в соответствии с заданными фильтрами | | Необходимые туры появились | Прошёл |
| Вопрос в чате | Сообщение успешно отправлено | | Сообщение успешно отправлено | Прошёл |
| Статистика сайта | Появится статистика сайта | | Статистика сайта появилась | Прошёл |
| Редактировать профиля пользователем | Профиль пользователя редактируется | | Профиль пользователя редактируется | Прошёл |
| Выход из системы пользователем | Пользовать выходит из системы | | Пользователь вышел из системы | Прошёл |

Таким образом, тестирование основного функционала приложения успешно пройдено. Весь заявленный функционал отвечает поставленным функциональным требованиям.

* 1. **Тестирование безопасности базы данных и веб-приложения**

Для обеспечения корректности работы программы, обрабатываются различные ошибки, возникающие в процессе работы. Данное программное средство использует подключение к базе данных, следовательно, неправильно введенные данные или же их отсутствие может повлечь за собой неработоспособность приложения.

В дипломном проекте организована валидация полей на случай неочевидного или неправильного поведения пользователя.

При авторизации обрабатываются вводимые данные пользователя на уровне клиентской части. При не введенных данных в поля авторизации, будет неактивна кнопка авторизации, а также при попытке ввода и удаление данных будут выведены сообщения об ошибках. На рисунке 4.1 продемонстрировано, что будет если нарушено одно из условий ограничения целостности при авторизации. В данном случае должна быть указана почта и пароль.

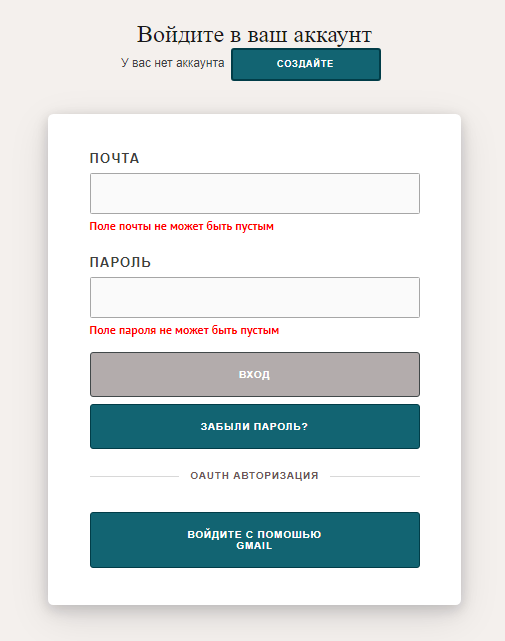


Рисунок 4.1 – Ошибки пустых полей и неактивная кнопка вход

Далее рассмотрим, что происходит при попытке регистрации пользователя с уже существующей почтой в базе данных. Для этого введем в поле почты уже зарегистрированного пользователя [*alexander.konopatsky@gmail.com*](mailto:alexander.konopatsky@gmail.com), представлено на рисунке 4.2.

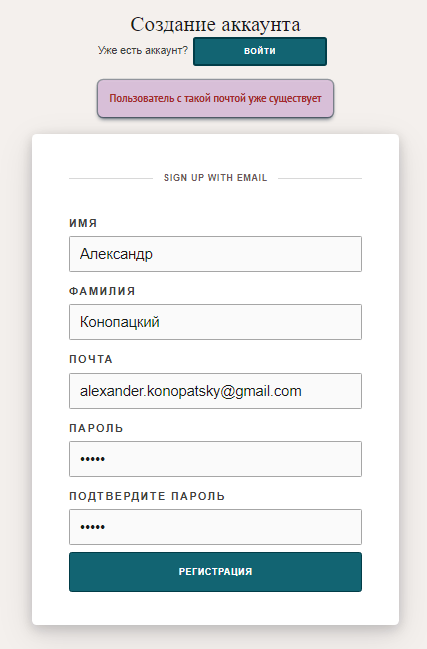


Рисунок 4.2 – Регистрация с существующей почтой

На рисунке 4.3 показано, что если неавторизованный пользователь (гость) попробует забронировать тур, то он получит предупреждение о том, что он не авторизован.

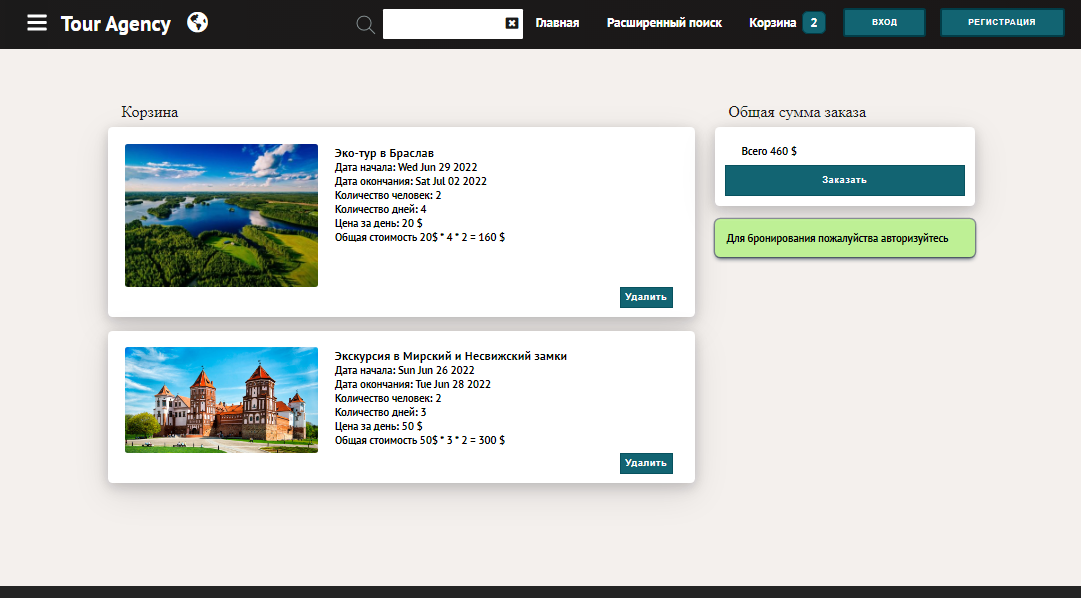


Рисунок 4.3 – Заказ туров не авторизованным пользователем

Далее рассмотрим случай, когда пользователь с ролью *user* пытается перейти по пути, предназначенному только для пользователей типа administrator. В данном случае приложение возвращает ошибку 403, которая представлена на рисунке 4.4.

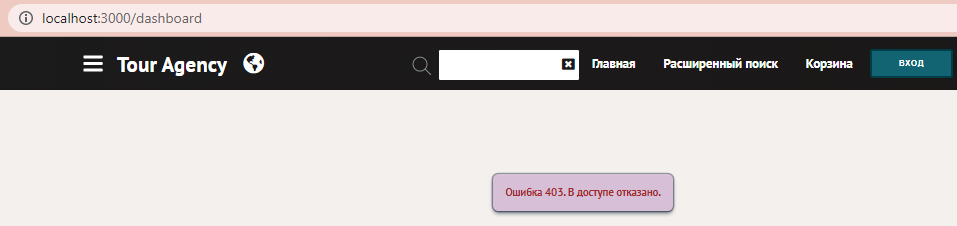


Рисунок 4.4 – Возврат ошибки 403

Если пользователь попытается перейти по несуществующему пути, приложение вернёт ему ошибку 404, которая представлена на рисунке 4.5.

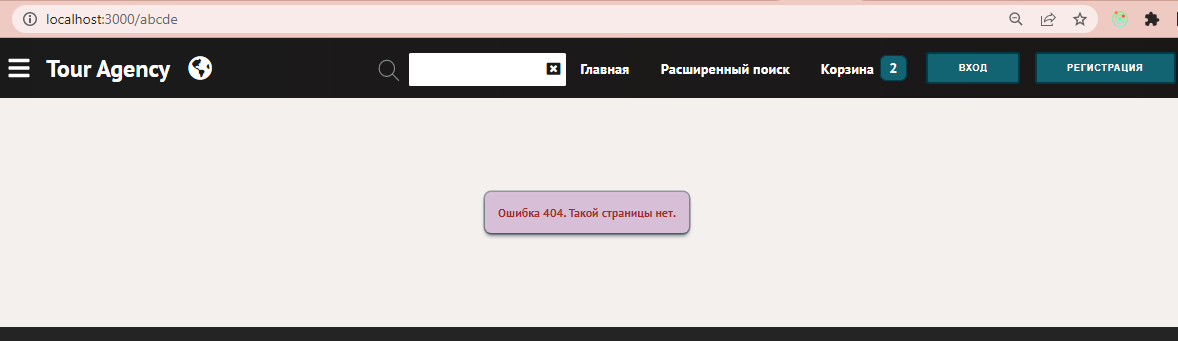


Рисунок 4.5 – Возврат ошибки 404

Далее рассмотрим случай, когда пользователь пробует оставить комментарий до заказа тура. В таком случае, будет выведено уведомление об ошибке, представлено на рисунке 4.6.



Рисунок 4.6 – Попытка написать комментарий без заказа тура

Если пользователь попытается отправить форму без оценки и комментария. То в таком случае, будет выведено уведомление об ошибке, которое представлено на рисунке 4.7.

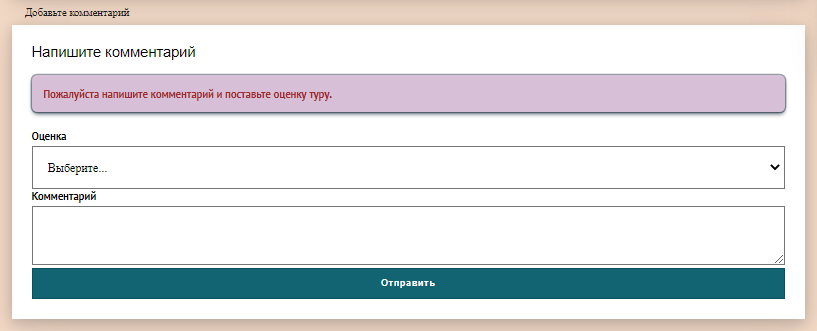


Рисунок 4.7 – Попытка отправить пустую форму комментария

Если пользователь напишет комментарий длиной более 1000 символов, то получит уведомление с соответствующей ошибкой. Уведомление представлено на рисунке 4.8.

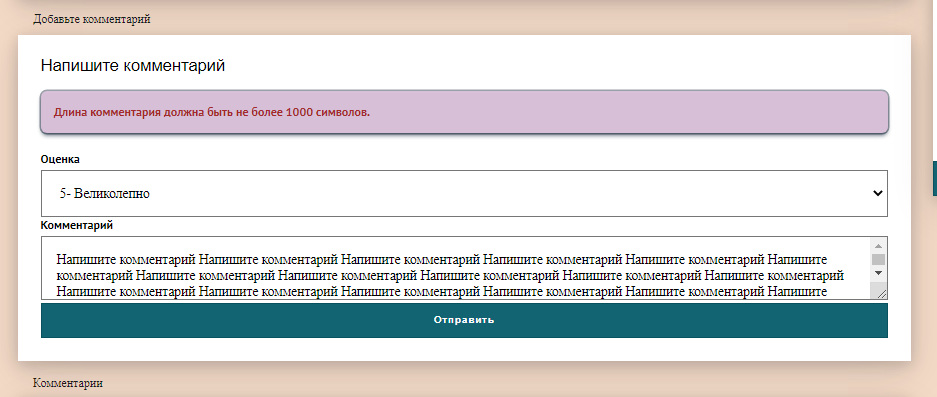


Рисунок 4.8 – Отправка слишком длинного комментария

* 1. **Выводы по разделу**

В данном разделе было проведено двадцать три ручных теста, в которых были рассмотрены различные случаи поведения конечного пользователя при использовании веб-сервиса. Из представленных в данном разделе тестов можно сделать вывод, что программное средство показывает стабильные результаты работы, а также весь функционал соответствует требованиям, работает корректно и выполняет поставленные задачи.

1. **Методика установки и использования**

ФИО

Подпись

Дата

Лист

1

БГТУ 05.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

Наркевич А. С.

Консульт.

Наркевич А. С.

Н. контр.

Макарчук М.. В.

.

Утв.

Пацей Н. В.

5 Методика установки и использования

Лит.

Листов

12

*74417039, 2022*

* 1. **Развёртывание системы**

Так как при разработке серверной части использовался *Node*.js*,* то приложение является кроссплатформенным, то есть может запускаться на различных операционных системах (*Windows*, *Linux*, *MacOS*). В том числе на мобильных устройствах. Кроссплатформенность обеспечивается благодаря использованию [высокоуровневых языков программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ.

Для запуска серверной части необходимо запустить команду *npm* *install*, которая установит все необходимые зависимости, и после этого выполнить команду *node* *server*, которая запустит веб-сервер.

Для сборки клиентской части необходимо запустить команду *npm* *start*, которая запускает сборку всех компонентов приложения в один *js* файл.

После этих действий приложение полностью готово к запуску.

* 1. **Руководство пользователя**
     1. **Иерархическая система ролей**

В приложении имеется три типа пользователя:

* гость;
* пользователь;
* администратор.

Функционал у данных типов пользователей имеет иерархическую структуру – это значит, что роль «пользователь» имеет функционал гостя и функционал, предназначенный для роли «пользователь», а роль «администратор» – функционал роли «пользователь», функционал роли «гость» и «администратор».

* + 1. **Основные возможности роли «Гость»**

Роль пользователя предназначена для неавторизованных пользователей. У данного типа пользователя есть следующие возможности:

* регистрация;
* авторизация через почту и пароль;
* авторизация через почту *gmail*;
* просмотр каталогов туров;
* просмотр туров;
* расширенный поиск туров
* добавление туров в корзину.

Для авторизации пользователь должен перейти на страницу авторизации. Ссылка на данную страницу имеется в правом верхнем углу под псевдонимом «Вход». Страница представлена на рисунке 5.1.

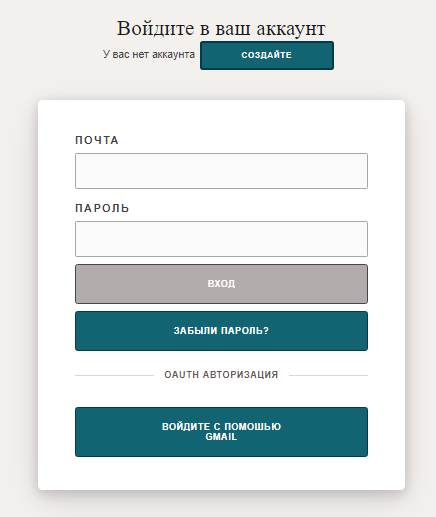


Рисунок 5.1 – Страница авторизации

Чтобы пройти процесс авторизации, пользователь должен быть зарегистрирован, либо может использовать вход через почту *gmail* (без необходимости доверять приложению логин, пароль и почту). Страница регистрации представлена на рисунке 5.2.

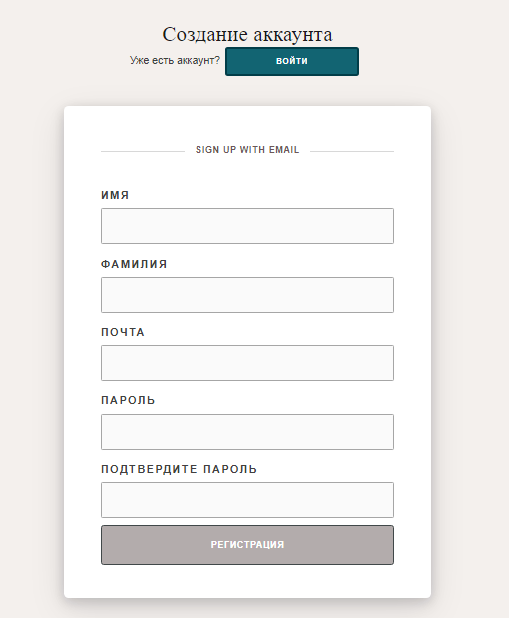


Рисунок 5.2 – Страница регистрации

На главной странице гость видит форму для поиска туров, с возможностью выбора города, диапазона дат и количества человек, а также несколько каталогов туров с лучшими оценками. Данная страница представлена на рисунке 5.3.

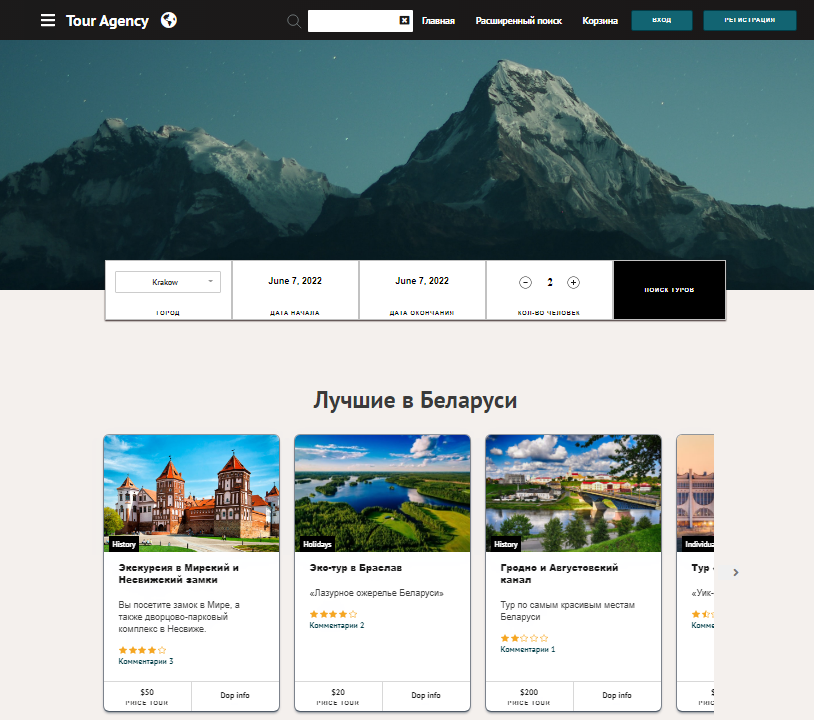


Рисунок 5.3 – Главная страница

Также гость может просматривать сами туры. В заголовке страницы указано название тура, короткое описание и количество звёзд, основанное на средней оценке пользователей. Ниже находится полное описание тура и календарь для выбора диапазона дат, в течении которых будет бронироваться тур. Данная страница представлена на рисунке 5.4.

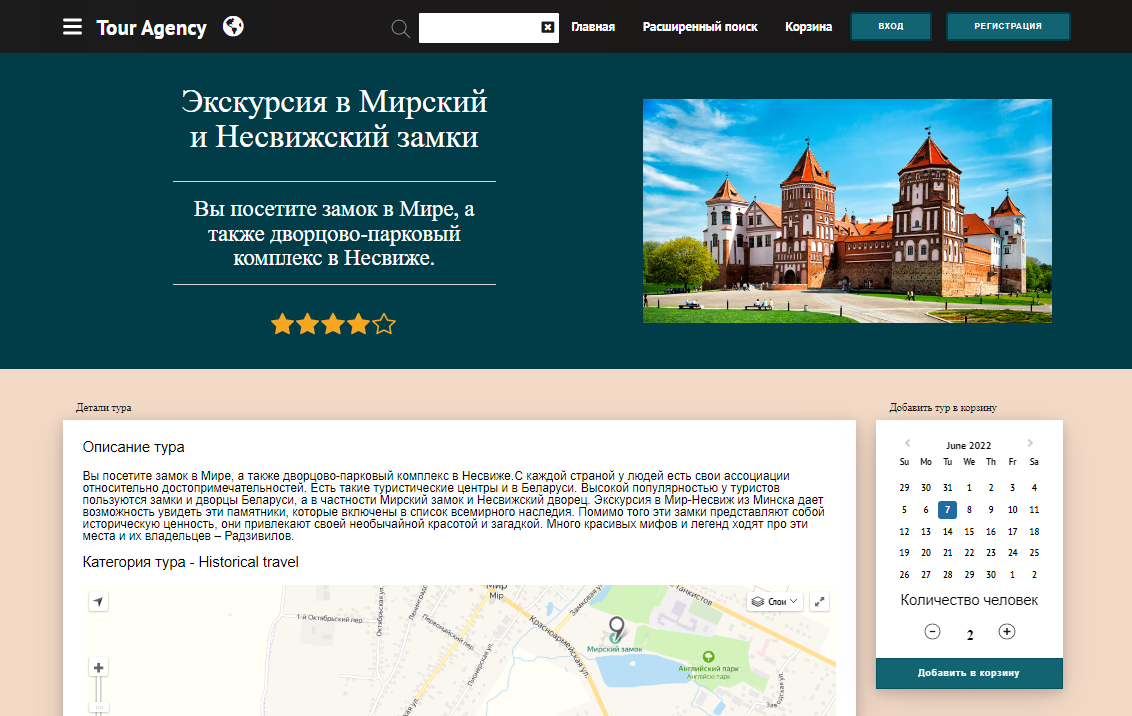


Рисунок 5.4 – Страница тура

Немного ниже на странице находится карта с основными достопримечательностями, галерея изображений и таблицы с доступными местами и датами бронирования. Данный функционал представлен на рисунке 5.5.

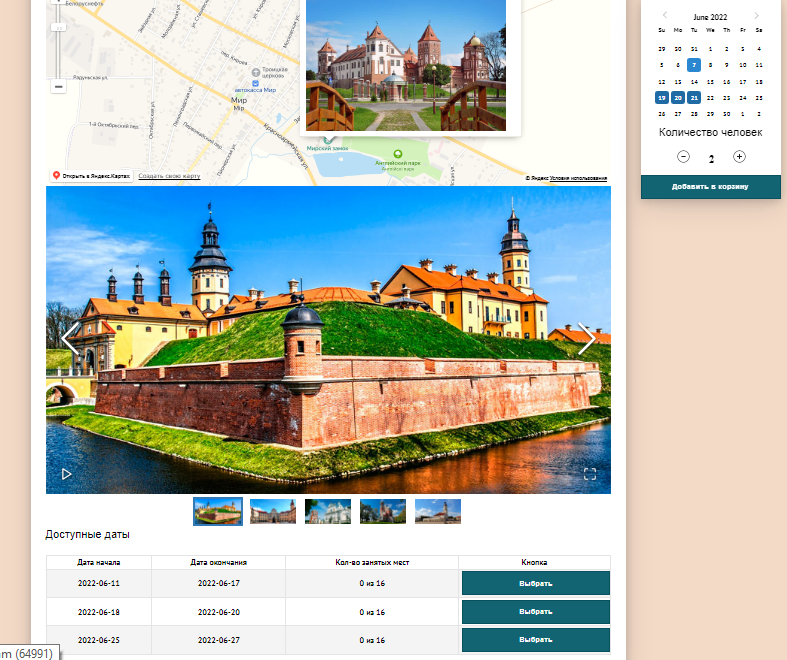


Рисунок 5.5 – Функционал достопримечательности на странице тура

Также гость может добавлять туры в корзину. У каждого тура есть цена за сутки, количество человек, дата начала и дата окончания, исходя из этих данных рассчитывается общая стоимость тура. Страница корзины представлена на рисунке 5.6.

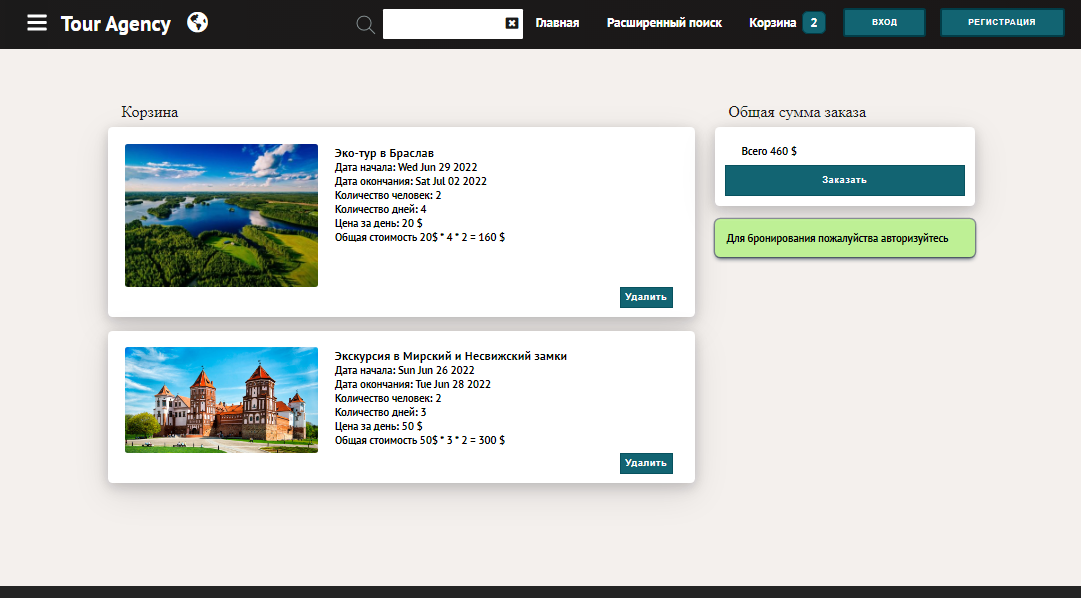


Рисунок 5.6 – Страница корзины гостя

Для быстрого поиска по категориям, странам и городам, гость может использовать боковую панель, которая представлена на рисунке 5.7.

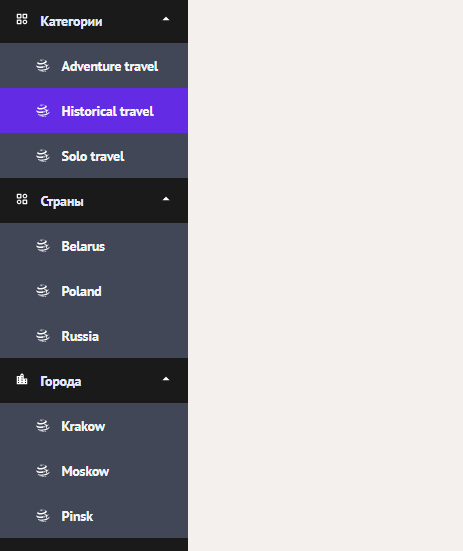


Рисунок 5.7 – Боковая панель поиска

Также гость может выполнять расширенный поиск по своим предпочтениям. Для этого необходимо в течении семи шагов указать интересующие параметры поиска. Страница расширенного поиска представлена на рисунке 5.8.

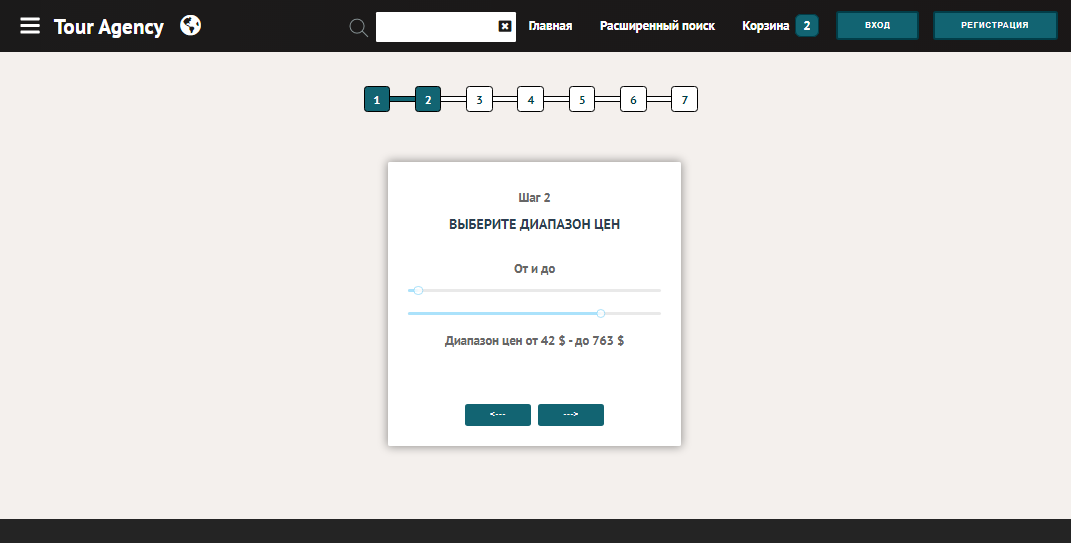


Рисунок 5.8 – Страница расширенного поиска

* + 1. **Основные возможность роли «Пользователь»**

Роль пользователя предназначена для авторизованных пользователей. У данного типа пользователя, помимо всех возможностей роли гость, есть следующие возможности:

* заказ тура;
* просмотр всех заказов;
* отмена заказа;
* просмотр и редактирование профиля;
* возможность задать вопрос в чате;
* рекомендации и история просмотров.

Как было сказано ранее, пользователь может заказать тур из страницы «Корзина». После бронирования пользователь окажется на странице с заказом, где может просмотреть заказ или отменить его. Данная страница с заказанными турами представлена на рисунке 5.9.

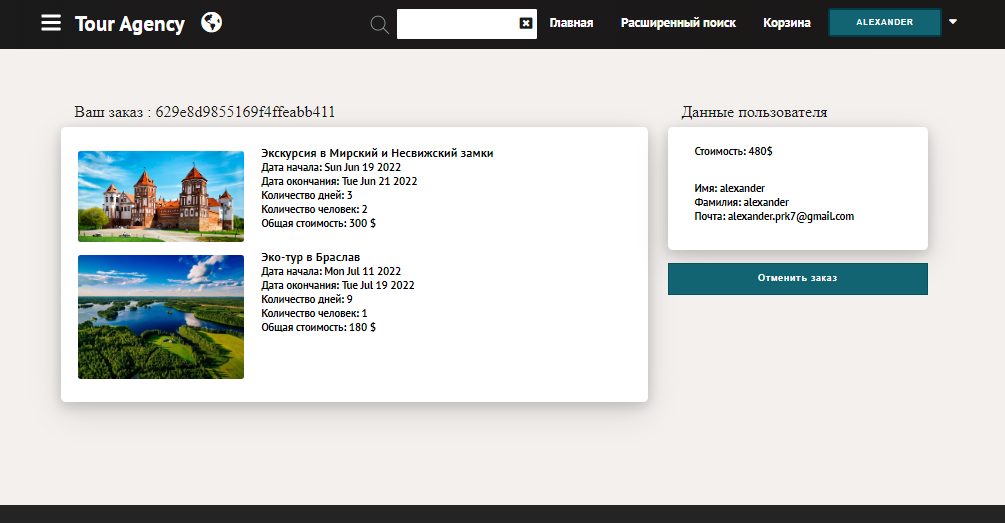


Рисунок 5.9 – Страница с заказанным туром

Также пользователь может просмотреть свою историю заказов. Ссылка на страницу находится в раскрывающемся меню в правом верхнем углу. Страница истории заказов представлена на рисунке 5.10.

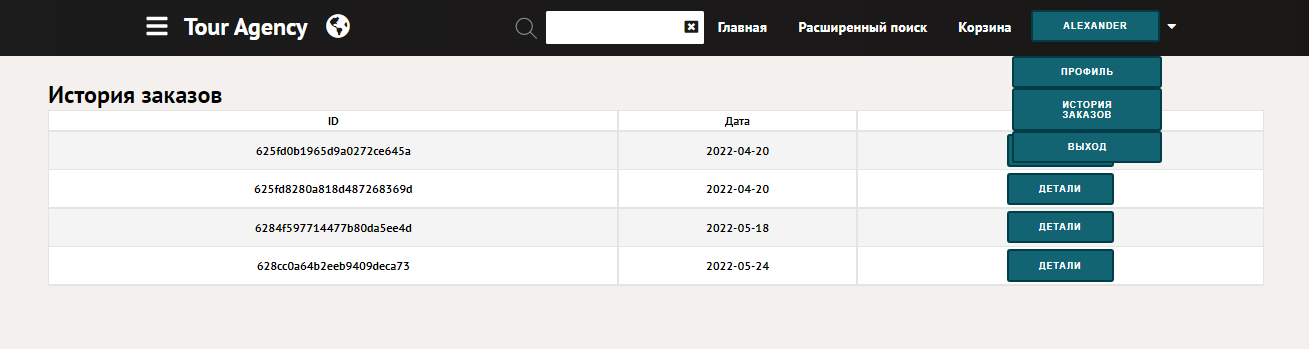
****

Рисунок 5.10 – Страница с историей заказов

Также пользователь может редактировать свой профиль. Страница профиля представлена на рисунке 5.11.

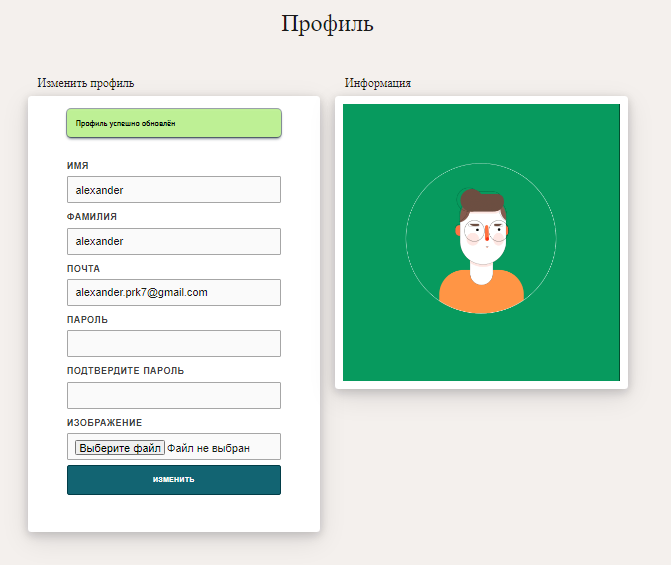


Рисунок 5.11 – Страница профиля

Также пользователь может обраться в поддержку и задать интересующий вопрос. В правом нижнем углу есть иконка чата, которая присутствует на всех страницах. Данный функционал представлен на рисунке 5.12.

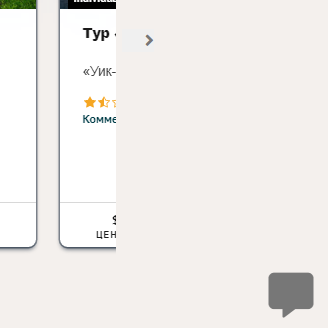


Рисунок 5.12 – Иконка чата

Для того, чтобы задать вопрос, необходимо нажать на иконку, откроется чат. После этого можно задавать вопрос. Данный функционал представлен на рисунке 5.13.

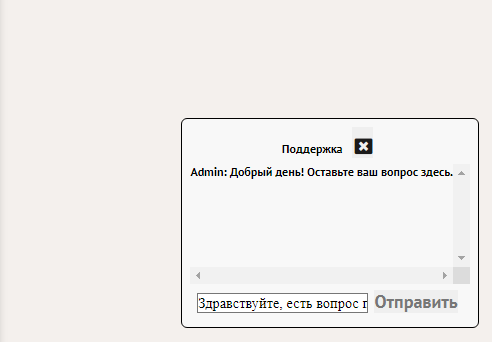


Рисунок 5.13 – Функционал чата

Также на главной странице у пользователя есть раздел с рекомендациями, основанных на предыдущих запросах пользователя, и история просмотров туров. Данный функционал представлен на рисунке 5.14.

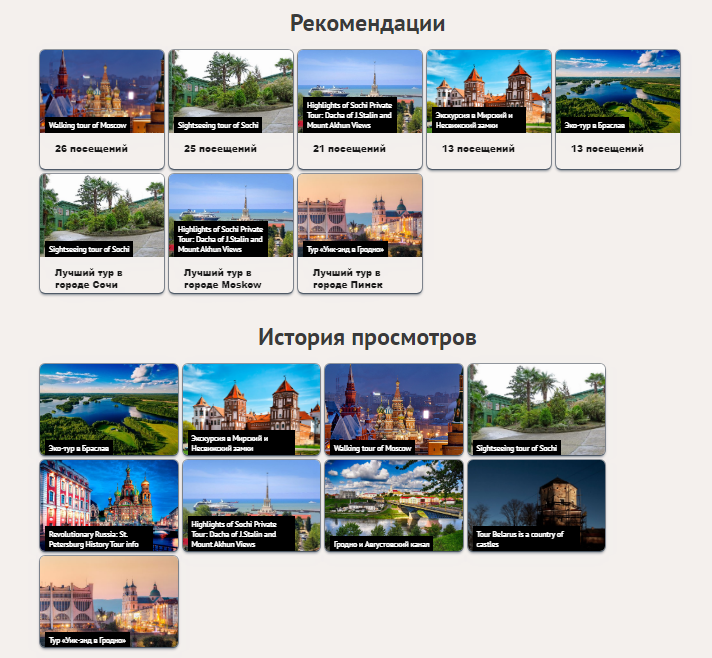


Рисунок 5.14 – Рекомендации и история просмотров пользователя

* + 1. **Основные возможности роли «Администратор»**

У пользователя с ролью администратора, помимо всех возможностей роли «гость» и роли «пользователь», есть следующие возможности:

* добавление, изменение, удаление туров и достопримечательностей;
* добавление, изменение, удаление категорий;
* добавление, изменение, удаление информации о странах;
* добавление, изменение, удаление информации о городах;
* одобрение, отключение и удаление комментарий;
* возможность отвечать на вопросы пользователей в чате;
* просмотр статистики приложения.

Как было сказано ранее, администратор может добавлять, изменять, удалять туры и достопримечательности. Для получения координат места достопримечательности, необходимо написать адрес в поле «Адрес места» и нажать кнопку «Получить координаты». Данная страница представлена на рисунке 5.15.

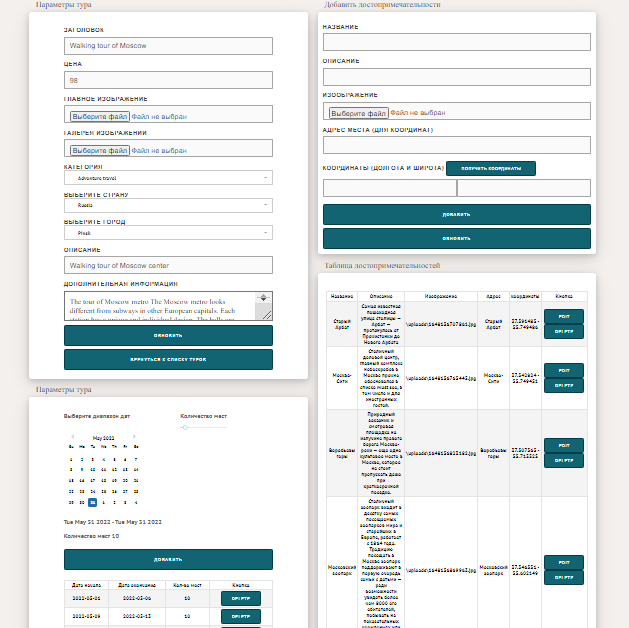


Рисунок 5.15 – Страница редактирования тура

Также администратор может одобрять и удалять комментарии пользователей. В верхнем окне находятся одобренные комментарии, в нижнем окне отключенные. Данный функционал представлен на рисунке 5.16.

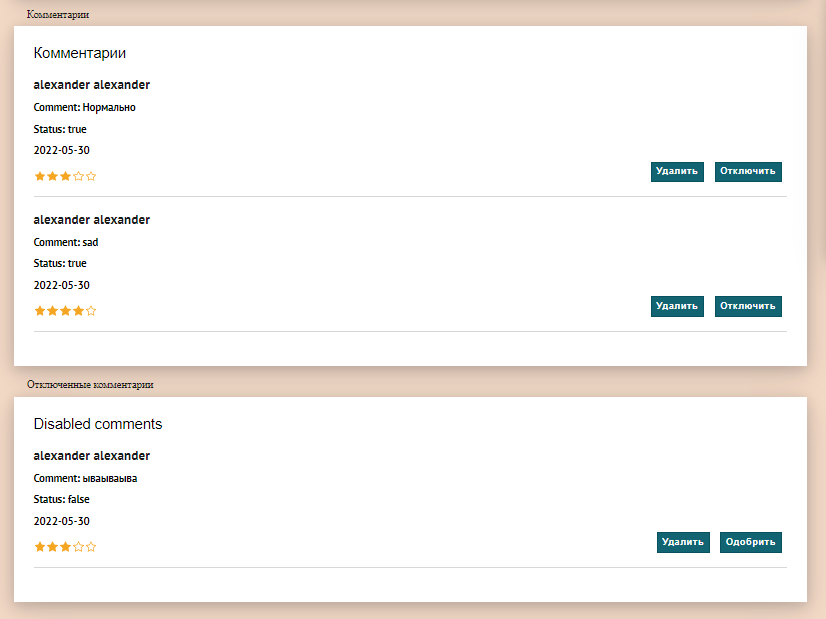
****

Рисунок 5.16 – Страница тура для одобрения и удаления комментариев

Также администратор может отвечать на вопросы пользователей в чате на странице «Поддержка». В левой части расположены активные чаты, зелёный цвет означает то, что пользователь в данный момент открыл чат и может начать писать сообщение, красный цвет означает пропущенное сообщение. Страница поддержки представлена на рисунке 5.17.

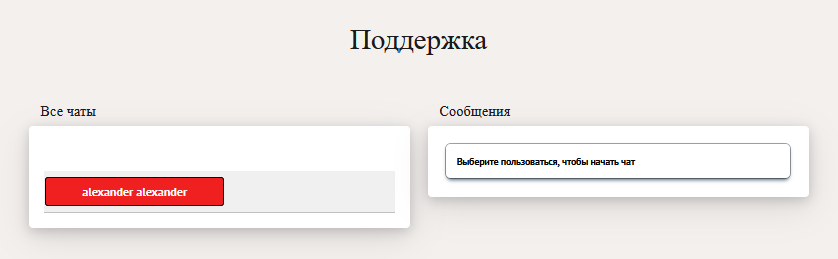
****

Рисунок 5.17 – Страница «Поддержка» для администратора

Для того, чтобы начать диалог с пользователем, необходимо нажать на кнопку с именем пользователя. После этого в правой части откроется диалог с загруженной историей переписки. Страница поддержки с активным чатом представлена на рисунке 5.18.

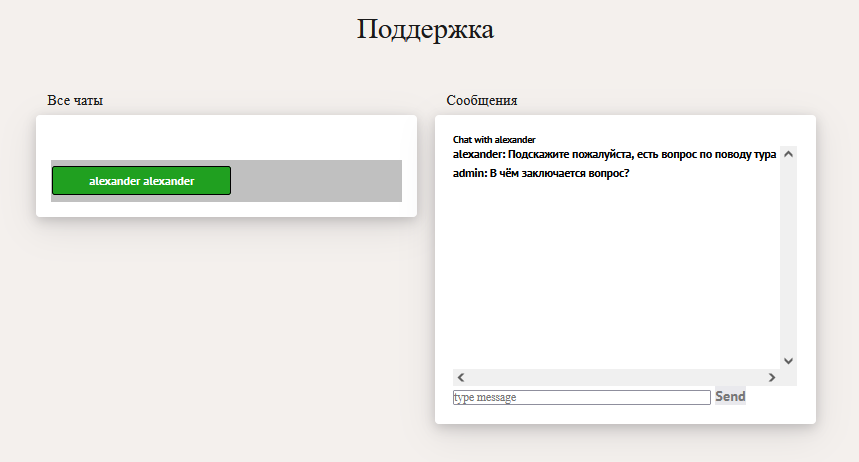


Рисунок 5.18 – Страница «Поддержка»

Также администратор может просматривать статистику приложения на странице «Панель администратора» за весь период работы приложения. Данная страница представлена на рисунке 5.19.

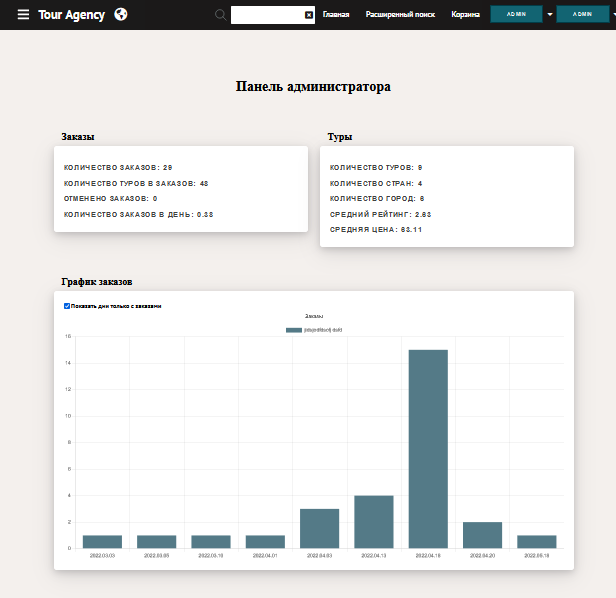


Рисунок 5.19 – Страница «Панель администратора»

Также на данной странице есть диаграммы с самыми популярными и самыми просматриваемыми турами. Данные диаграммы представлены на рисунке 5.20.



Рисунок 5.20 – Страница «Панель администратора».

* 1. **Выводы по разделу**

В данном разделе было описано руководство пользователя, где описан основной функционал пользователей всех ролей.

Был подробно рассмотрен процесс взаимодействия различных ролей с системой, рассмотрены возможные пути использования приложения, описан существующий функционал приложения, а также описан процесс запуска проекта с использованием некоторых операционных систем.

1. **Технико-экономическое обоснование дипломного проекта**

ФИО

Подпись

Дата

Лист

1

БГТУ 06.00.ПЗ

Разраб.

*Конопацкий А.А.*

Пров.

*Наркевич А. С.*

.

Консульт.

*Наркевич А. С.*

Н. контр.

Макарчук М. В.

Утв.

Пацей Н. В.

6 Технико-экономическое обоснование дипломного проекта

Лит.

Листов

12

*74417039, 2022*

* 1. **Общая характеристика разрабатываемого программного средства**

Основной целью данного раздела является экономическое обоснование целесообразности разработки веб-приложения, представленного в дипломном проекте. Для выполнения поставленной цели будут определены экономические показатели, в том числе себестоимость продукта.

В этом разделе пояснительной записки проводится расчет затрат на всех стадиях разработки, а также анализ экономического эффекта в связи с использованием данного веб-приложения.

Разработанный программный продукт позволяет пользователям найти подходящий тур и выполнить бронирование данного тура. Администратор, в свою очередь, может создавать новые туры, наполнять их информацией, одобрять и удалять комментарии пользователей, а также отвечать в чате на вопросы пользователей.

Способ монетизации – продажа разработанного программного средства и смежных прав на него заказчику.

В рамках данного раздела необходимо определить затраты, произведенные на всех стадиях разработки описанного программного средства.

* 1. **Исходные данные для проведения расчётов**

Созданная система программ не является принципиально новым программным средством, ввиду наличия большого количества аналогов. Она принадлежит ко второй группе программ по степени сложности. Источниками исходных данных для расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета стоимости разработки приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Численность разработчиков | чел. | Чр | 1 |
| Норматив дополнительной заработной платы | % | Нд.з. | 10 |
| Ставка отчислений в Фонд социальной защиты населения | % | Нф.с.з.н. | 34,6 |

Окончание таблицы 6.1

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателя | Единица измерения | Условные обозначения | Норматив |
| Ставка отчислений по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний | % | Нб.г.с. | 0,3 |
| Стоимость одного машино-часа | руб. | См.ч. | 0,05 |
| Норматив прочих затрат | % | Нп.з. | 24,5 |
| Норматив общепроизводственных и общехозяйственных расходов | % | Нобп, обх | 178 |
| Норматив расходов на сопровождение и адаптацию | % | Нр.с.а. | 17 |
| Уровень рентабельности | % | Урент. | 30 |
| Ставка НДС | % | Нндс. | 20 |
| Коэффициент изменения скорости обработки информации | ед. | Кск. | 0,6 |
| Тарифная ставка I разряда в организации за месяц | руб. | ОТI | 780 |
| Тарифный коэффициент | ед. | Кт. | 3,04 |
| Коэффициент естественных потерь рабочего времени | ед. | Кп. | 1,1 |
| Коэффициент премирования | ед. | Кпр. | 1 |
| Норматив расхода машинного времени за отладку 100 строк исходного кода | маш. часов | Нм.в. | 12 |

Далее эти данные понадобятся для расчета всех необходимых параметров в процессе экономического анализа приложения.

* 1. **Методика обоснования цены**

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

* снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
* сокращения расходов на оплату машинного времени и других ресурсов на отладку программных средств;
* снижения расходов на материалы;
* ускорение ввода в эксплуатацию новых систем;
* улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка веб-приложения у разработчиков предполагает определение затрат, что включает следующие статьи:

* заработная плата исполнителей – основная и дополнительная;
* отчисления в Фонд социальной защиты населения и обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
* расходы на материалы и комплектующие;
* расходы на спецоборудование;
* расходы на оплату машинного времени;
* общепроизводственные и общехозяйственные расходы;
* расходы, связанные с установкой и адаптацией (реализационные или коммерческие расходы).

На основании затрат рассчитывается себестоимость и отпускная цена разработанного веб-приложения.

* + 1. **Определение объёма программного средства**

На основании информации о функциях, приведенной в таблице 6.3, рассчитывается общий объем программного средства *V*o, условных машинных команд, по формуле 6.1.

, (6.1)

где *Vi* – объем i-ой функции ПС, условных машинных команд;

*n* – общее число функций.

Ниже приведена таблица с информацией о затратах рабочего времени на разработку программного средства.

Таблица 6.2 – Содержание и объем функций на разрабатываемое ПС

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, дней |
| Инициализация клиентской и серверной частей | 1 |
| Инициализация базы данных, настройка контейнеров | 2 |

Окончание таблицы 6.2

|  |  |
| --- | --- |
| Содержание работ | Затраты рабочего времени, дней |
| Написание логики аутентификации и регистрации | 6 |
| Создание ролей пользователей | 2 |
| Написание логики связанной с турами | 9 |
| Написание логики связанной с достопримечательностями | 4 |
| Написание логики связанной с расширенным поиском | 5 |
| Написание логики связанной с комментариями | 7 |
| Написание логики связанной с бронированием туров | 6 |
| Написание логики связанной с чатом | 9 |
| Написание логики связанной с анализом и статистикой | 7 |
| Тестирование приложения | 4 |
| Всего | 62 |

По данным, приведенным в методическом пособии, в разработанном приложении присутствуют функции, приведенные в таблице 6.2.

Таблица 6.3 – Содержание и объем функций на разрабатываемое ПС

|  |  |  |
| --- | --- | --- |
| № функции | Содержание функции | Объем, условных машинных команд |
| 111 | Управление вводом/выводом | 2 400 |
| 203 | Формирование базы данных | 2 180 |
| 204 | Обработка наборов и записей базы данных | 2 670 |
| 208 | Организация поиска в базе данных | 5 480 |
| 210 | Загрузка базы данных | 2 780 |
| 507 | Обеспечение интерфейса между компонентами | 970 |
| 601 | Отладка прикладных программ в интерактивном режиме | 4 300 |
| 703 | Расчет показателей | 460 |
| 707 | Графический вывод результатов | 480 |
|  | Итого | 21 720 |

Исходя из данных таблицы 6.3, можно рассчитать объем программного сред­ства, разработанного в процессе дипломного проектирования:

*V*o 2 400 + 2 180 + 2 670 + 5 480 + 2780 +

+ 970 + 4 300 + 460 + 480 = 21 720 (маш. команд).

В связи с достаточно быстрым изменением вычислительной техники рекомендуется определить скорректированный объем функций **, условных машинных команд по формуле 6.2.

** = *Vo*·*Кск*, (6.2)

где *Кск*– коэффициент изменения скорости обработки информации.

Коэффициент изменения скорости обработки информации равен 0,6 – по исходным данным, приведенным в таблице 6.1.

* =* 21 720 · 0,6 = 13 032 (маш. команд).

Таким образом, скорректированный объем функций условных машинных командравен 13032 условных машинных команд.

* + 1. **Основная заработная плата**

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере веб-программирования на *JavaScript*. В итоге было установлено, что средняя месячная заработная плата на позиции *junior/middle* составляет 1900 рублей.

Основная заработная плата рассчитывается по формуле 6.4.

|  |  |
| --- | --- |
| Соз = Tраз ⋅ Краз⋅ Сзп, | (6.4) |

где Соз– основная заработная плата, руб.;

Траз – время разработки, месяцев;

Краз – количество разработчиков, человек;

Сзп – средняя месячная заработная плата.

Время разработки будет рассчитываться по формуле 6.5.

|  |  |
| --- | --- |
| Траз = Драз / 21, | (6.5) |

где Траз – время разработки, месяцев;

Драз – количество дней разработки, дней.

Проект разрабатывался одним человеком на протяжении трех месяцев. Расчеты основной заработной платы и времени разработки представлены ниже:

Траз = 62 / 21 = 2,95 месяцев,

Соз = 2,95 ⋅ 1 ⋅ 1900 = 5605 руб.

* + 1. **Дополнительная заработная плата**

Законодательство о труде предусматривает наличие выплат, определяющиеся по нормативу в процентах к основной заработной плате по формуле 6.6.

, (6.6)

где Соз – основная заработная плата, руб.;

Ндз – норматив дополнительной заработной платы, %.

 руб.

Дополнительная заработная плата включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах по отношению к основной заработной плате.

* + 1. **Расчёт отчислений на социальные цели**

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 6.7.

|  |  |
| --- | --- |
|  | (6.7) |

где Соз – основная заработная плата, руб.;

Сдз – дополнительная заработная плата, труб.;

Нфсзн – норматив отчислений в Фонд социальной защиты населения, %.

Исходя из полученных данных высчитаем отчисления в Фонд социальной защиты наcеления (ФСЗН):

руб.

Полученные выше значения будут использованы при подсчете заработной платы программиста с отчислениями.

* + 1. **Расходы на материалы**

Расходы на материалы определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода НМ на уточненный объем веб-приложения Vo/. Сумма затрат на материалы рассчитывается по формуле 6.8.

 (6.8)

где Нм – норма расхода материалов в расчете на сто строк исходного кода программного средства, руб;

V’о – скорректированный объем функций, условных машинных команд.

Так как программное средство решает задачи поиска в базе данных и обработки строк, сопоставимые по ресурсоемкости с задачами расчетного характера, то Нм = 0,46 руб. (по данным, приведенным в приложении 2 таблице П 2.10 «Оценка значений среднего расхода материалов на разработку и отладку 100 строк кода применения веб-приложения» методического пособия [33])

См = 0,46 ⋅ 13032 / 100 = 59,95 руб.

Сумма расходов на материалы была вычислена на основе данных приведенных в таблице 6.1 данного дипломного проектирования.

* + 1. **Расходы на оплату машинного времени**

Сумма расходов на оплату машинного времени Смв определяется как произведение стоимости одного машино-часа Смч на уточненный объем программного средства ** и на норматив расхода машинного времени на отладку ста строк исходного кода Нмв и рассчитывается по формуле 6.9.

 (6.9)

где Смв – сумма расходов на оплату машинного времени, руб.;

Цм – цена одного машино-часа, руб.;

Нмв – норматив расхода машинного времени на отладку 100 строк программного кода, машино-часов;

Vo/ – уточненный объем веб-приложения, условных машино-команд.

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 0,05, можно определить сумму расходов на оплату машинного времени:

Смв = 0,05 ⋅ 13032 ⋅ 12/ 100 = 78,19 руб.

Расходы на оплату машинного времени, необходимого для разработки и отладки разрабатываемого веб-приложения,равны 78,19 рублей.

* + 1. **Расчет прочих прямых затрат**

Расходы на конкретное программное средство Спз включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы и рассчитываются по формуле 6.10.

|  |  |
| --- | --- |
|  | (6.10) |

где Нпз – норматив прочих затрат в целом по организации, %.

= 5605 ⋅ 24,5/100 = 1373,2 руб.

Таким образом, прочие прямые затраты при разработке веб-приложенияравны 1373,2 рублей.

* + 1. **Расчет общепроизводственных и общехозяйственных расходов**

Сумма общепроизводственных, общехозяйственных расходов Собп,обх  – произведение основной заработной платы исполнителей на конкретное программное средство Соз на норматив накладных расходов в целом по организации Нобп,обх. по формуле 6.11.

|  |  |  |
| --- | --- | --- |
|  |  | (6.11) |

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

Собп,обх = ⋅ 178 / 100 =  руб.

Получим, что накладные расходы, необходимые для разработки веб-приложения,равны 9976,9 рублей.

* + 1. **Расчет суммы расходов на разработку программного средства**

Сумма расходов на разработку программного средства Ср определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов.

Ср = Соз + Сдз + Сфсзн + См + Смв + Спз + Собп,обх, (6.12)

Ср =5605 + 560,5 + 2133,26 + 59,95+ 78,19 + 1373,2 + 9976,9 =19787 руб.

Таким образом, получим сумму всех расходов на разработку веб-приложенияравной 19787 рублей.

* + 1. **Расходы на сопровождение и адаптацию**

Сумма расходов на сопровождение и адаптацию программного средства Срса определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию Нрса.

|  |  |  |
| --- | --- | --- |
|  |  | (6.13) |

где Ср.с.а – сумма расходов на сопровождение и адаптацию ПС, руб.;

Ср – общая сумма расходов на разработку ПС, руб.;

Нр.с.а – норматив расходов на сопровождение и адаптацию, %.

Основываясь на исходные данные, расположенные в таблице 6.1 и формулу 6.13, норматив расходов на сопровождение и адаптацию Нрса равен 10%. Сумма расходов на сопровождение и адаптацию ПС составляет:

руб.

Получим, что расходы, необходимые на сопровождение и адаптацию веб-приложения,равны рубля.

* + 1. **Расчет общей суммы расходов**

Общая сумма расходов на разработку с затратами на сопровождение и адаптацию Сп определяется по формуле 6.14.

|  |  |  |
| --- | --- | --- |
|  | , | (6.14) |

Сп = 19787+  = 21765,7 руб.

Получим, что общая сумма расходов веб-приложения, выполняющего роль туристического агентства, равна 21765,7 рубля.

* + 1. **Определение цены, оценка эффективности**

Отпускная цена рассчитывается на основании цены разработчика, которая формируется на основе показателя рентабельности продукции. Рентабельность и прибыль по создаваемому программному средству определяются исходя из результатов анализа рыночных условий, переговоров с заказчиком (потребителем) и согласования с ним отпускной цены, включающей дополнительно налог на добавленную стоимость.

Прибыль от реализации программного средства вычисляется по формуле 6.15.

|  |  |  |
| --- | --- | --- |
|  |  | (6.15) |

где Урент – уровень рентабельности, %;

Сп – полная себестоимость программного средства, руб.

Цена разработчика программного средства без налогов находится по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.16) |

Сумма налога на добавленную стоимость рассчитывается из соотношения:

|  |  |  |
| --- | --- | --- |
|  |  | (6.17) |

где Цр – цена разработчика программного средства, руб.;

Нндс – ставка НДС, %.

Планируемая отпускная цена с НДС вычисляется по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (6.18) |

Исходя из вышеописанных данных рассчитаем прибыль от реализации программного средства, цену разработчика без налогов, сумму налогов на добавленную стоимость, а также планируемую отпускную цену с учетом НДС.

Ппс = 21765,7  / 100 = 6529,71 руб.,

Цр = 21765,7 + 6529,71 = 28295,41 руб.,

НДС = 28295,41  / 100  = 5695,08 руб.,

Цс НДС = 28295,41 + 5695,08 = 33990,49 руб.,

Для анализа стоимости разработки необходимо рассмотреть стоимость разработки приложений, обладающих аналогичным функционалом. Для этого воспользуемся помощью трех сайтов-агрегаторов услуг по разработке программного обеспечения.

Первым ресурсом является сайт *estimatemyapp.com*. Выбрав все необходимые параметры, выходит, что на разработку схожего ресурса в среднем должен уйти 93 дня разработки, а также стоимость в 51300 $. Наглядная демонстрация результатов работы ресурса представлена на рисунке 6.1.



Рисунок 6.1 – Результат работы сайта *estimatemyapp.com*

Вторым ресурсом является сайт *ios-lab.ru*. Этот ресурс работает по аналогии с предыдущим, за исключением того, что итоговая цена будет представлена в российских рублях. На рисунке 6.2 представлен результат работы сайта-агрегатора:

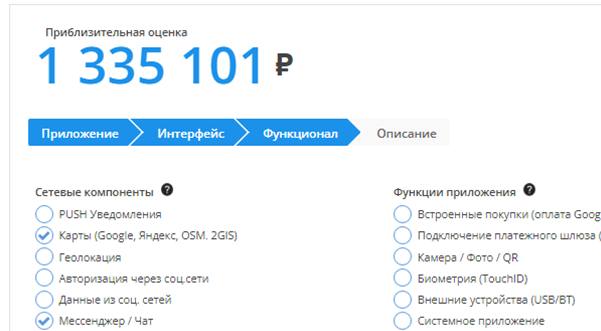


Рисунок 6.2 – Результат работы сайта *ios-lab.ru*

Как видно из результатов, примерная сумма похожего ресурса обойдется в 1330000 рублей, что в пересчете на белорусские рубли составляет 71800 рублей.

Последним ресурсом является сайт *ios-lab.ru*. Выбрав все необходимые компоненты системы, приблизительная стоимость проекта составляет 1955000 российский рубль, что в пересчете составляет 105000 белорусских рублей. Ниже на рисунке 6.3 приведен результат работы сайта.

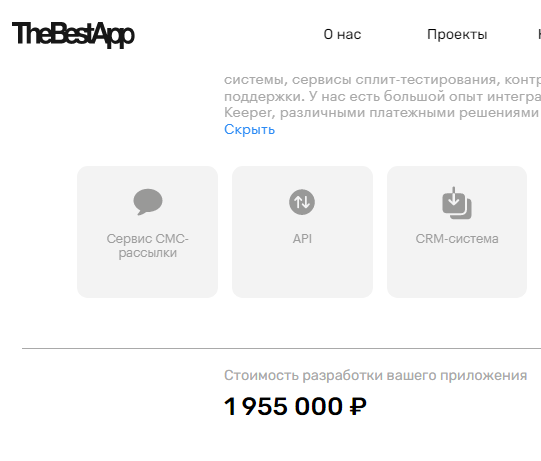


Рисунок 6.3 – Результат работы сайта *thebestapp.ru*

Как видно из результатов, отпускная цена на сайтах-агрегаторах услуг по разработке программного обеспечения не сильно отличается друг от друга.

* 1. **Вывод по разделу**

В таблице 6.4 представлены результаты расчетов для основных показателей данного раздела.

Таблица 6.4 – Результаты расчетов

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Время разработки, мес. | 2,95 |
| Количество программистов, чел. | 1 |
| Основная заработная плата, руб. | 5 605 |
| Сумма отчислений на социальные цели, руб. | 2 133,26 |
| Сумма расходов на материалы, руб. | 59,95 |
| Расходы на оплату машинного времени, руб. | 78,19 |

Окончание таблицы 6.4

|  |  |
| --- | --- |
| Наименование показателя | Значение |
| Прочие прямые затраты, руб. | 1 373,2 |
| Общепроизводственные и общехозяйственные расходу, руб. | 9 976,9 |
| Сумма расходов на разработку программного средства, руб. | 19 787 |
| Расходы на сопровождение и адаптацию ПС, руб. |  |
| Обща сумма расходов, руб. | 21 765,7 |
| Сумма налога на добавленную стоимость, руб. | 5 695,08 |
| Отпускная цена с НДС, руб. | 33 990,49 |
| Прибыль от реализации, руб. | 6 529,71 |

Исходя из совокупности всех вышеперечисленных факторов, определим, что разработка данного веб-приложения несет положительный экономический эффект.

Необходимость разработки программного средства, обусловлена повышением интереса к туризму у людей, так как за последние десятилетия туризм стал одним из самых динамичных и быстрорастущих секторов мировой экономики. Разработка вышеописанного программного средства, осуществлялась с целью повышения эффективности работы туристического агентства.

Разработка программного средства, осуществляемая одним программистом в течении трех месяцев, при заданных условиях обойдется компании в 33990,49 руб. Реализации данного программного средства принесет прибыль компании в сумме 6529,71 рублей. Цена за разработанный продукт ниже рыночных цен на продукцию, обладающих аналогичным функционалом, которые были рассмотрены ранее.

# **Заключение**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А .*

Пров.

Наркевич А. С.

Консульт.

Н. контр.

Макарчук М. В.

.

Утв.

Пацей Н. В.

Заключение

Лит.

Листов

1

*74417039, 2022*

В ходе выполнения дипломного проектирования были рассмотрены основные аналоги программного средства, проанализированы их преимущества и недостатки, а также просмотрены и изучены их возможности. Произведен обзор теоретических материалов по теме дипломного проекта и обзор аналогов по теме дипломного проекта.

Были представлены основные технические средства, которые использовались при разработке. Дана их краткая характеристика, возможности и достоинства.

Описана архитектура приложения, а также принципы взаимодействия между компонентами приложения. Особенности реализации *Node*.*js* сервера и взаимодействие с клиентом. Были описаны принципы и подходы, применяемые для взаимодействия серверной части приложения с базой данных.

Выполнено технико-экономическое обоснование разработки данного программного средства, рассчитана экономическая эффективность использования данного программного средства. На основании полученных данных можно сделать вывод, что разработанный проект является экономически выгодным.

При написании руководства для всех пользователей системы были предусмотрены и подробно описаны основные возможные действия пользователей.

Результатом проделанной работы стало веб-приложение для поиска и бронирования туров. Разработанным веб-приложением могут пользоваться несколько типов пользователей: гость, пользователей и администратор. Каждый тип пользователей обладает определенным функционалом. Данные приложения хранятся в базе данных, которая была спроектирована до разработки программного средства.

В рамках дипломного проекта было разработано веб-приложение «Туристическое агентство», которое соответствует предъявленным требованиям и поставленным задачам, а также было приведено руководство пользователя для разработанного веб-приложения.

Материалы, используемые в дипломном проектировании, были представлены на научно-технической конференции учащихся, студентов и магистрантов БГТУ, проводимой в УО «Белорусский государственный технологический университет» в 2022 году.

# **Список используемых источников**

ФИО

Подпись

Дата

Лист

1

БГТУ 00.00.ПЗ

Разраб.

*Конопацкий А.А .*

Пров.

Наркевич А. С.

Консульт.

Н. контр.

Макарчук М. В.

.

Утв.

Пацей Н. В.

Список использованных источников

Лит.

Листов

1

*74417039, 2022*

1. Веб-приложение [Электронный ресурс] / Википедия. – Режим доступа: https://ru.wikipedia.org/wiki/web. – Дата доступа: 19.03.2022
2. Сайт *Tez-tour* [Электронный ресурс]. – Режим доступа: https://www.tez-tour.com/. – Дата доступа 16.03.2022.
3. Сайт Tury.by [Электронный ресурс]. – Режим доступа: https://tury.by/. – Дата доступа 18.03.2022.
4. Сайт Tursales.by [Электронный ресурс]. – Режим доступа: https://tursales.by/. – Дата доступа 19.03.2021.
5. Сайт Peopletravel.by [Электронный ресурс]. – Режим доступа: https://peopletravel.by/. – Дата доступа 20.03.2021.
6. Руководство по языку JavaScript [Электронный ресурс] / learn.javascript.ru. – Режим доступа: https://learn.javascript.ru. – Дата доступа: 09.03.2022.
7. Node.js [Электронный ресурс] / nodejs.org. – Режим доступа: https://nodejs.org/en/docs/ . – Дата доступа: 05.02.2022.
8. Express [Электронный ресурс] / expressjs.com. – Режим доступа: https://expressjs.com/ru/guide/routing.html. – Дата доступа: 08.03.2022
9. Socket.io [Электронный ресурс] / socket.io. – Режим доступа <https://socket.io/docs/v4/how-it-works/>. – Дата доступа 07.02.2022.
10. React.js [Электронный ресурс] / ru.reactjs.org. – Режим доступа https://ru.reactjs.org/docs/getting-started.html. – Дата доступа: 05.02.2022.
11. Redux [Электронный ресурс] / redux.js.org. – Режим доступа https://redux.js.org/tutorials/essentials/part-1-overview-concepts. – Дата доступа: 05.02.2022.
12. MongoDB [Электронный ресурс]. – Режим доступа https://www.mongodb.com/docs/. – Дата доступа: 13.03.2022.
13. Mongoose [Электронный ресурс]. – Режим доступа https://mongoosejs.com/docs/. – Дата доступа: 19.02.2022.

ПРИЛОЖЕНИЕ А

Схема архитектуры приложения

ПРИЛОЖЕНИЕ Б

Диаграмма вариантов использования

ПРИЛОЖЕНИЕ В

Логическая структура базы данных

ПРИЛОЖЕНИЕ Г

Блок-схема алгоритма бронирования тура

ПРИЛОЖЕНИЕ Д

Диаграмма классов

ПРИЛОЖЕНИЕ Е

Диаграмма последовательности работы чата

ПРИЛОЖЕНИЕ Ж

Скриншот работы веб-приложения

ПРИЛОЖЕНИЕ И

Листинг роутера туров

|  |
| --- |
| const express = require('express')  const Tour = require('../models/tourModel')  const Category = require('../models/categoryModel')  const Order = require('../models/orderModel')  const City = require('../models/cityModel')  const Country = require('../models/countryModel')  const User = require('../models/userModel')  const tourRouter = express.Router()  const { isAuth, isAdmin } = require('../middleware/utils')  const ObjectId = require('mongodb').ObjectID;  tourRouter.get('/cityInTheCountry', async (req, res) => {  const country = req.query.country  let countryId = await Country.findOne({countryName : country}).select('\_id')  const ddd = await City.find({country : countryId.\_id}).select('cityName')  let countryList = []  ddd.forEach(item => {  countryList.push(item.cityName)  })  res.send(countryList)    })  tourRouter.post('/advancedSearchPage', async (req, res) => {    let tours = await Tour  .find({  price: {$gt : req.body.sliderValuePriceFrom, $lt: req.body.sliderValuePriceTo},  rating: {$gt : req.body.ratingValue},  country: req.body.countryValue  })  .populate({path: 'cityT', match : { cityName : req.body.cityValue}})  .populate({path: 'categoryS', match : { categoryName : req.body.categoryValue}})  .select('\_id title image category label desc additionalInfo price rating numReviews country categoryS cityT')  tours = tours.filter(tour => {  if (tour.cityT && tour.categoryS) return tour  })  console.log(tours)  res.send({tours})  })  tourRouter.get('/', async (req, res) => {  const pageSize = 3  const currentPage = Number(req.query.page) || 1  const priceFrom = req.query.priceFrom || 0  const priceTo = req.query.priceTo || 100000  const startDate = req.query.startDate  const endDate = req.query.endDate  const rating = req.query.rating || -1  const category = req.query.category  const country = req.query.country  const city = req.query.city  const categoryFilter = category ? { category } : {}  const countryFilter = country ? { country } : {}  const cityFilter = city ? { city } : {}  const ratingFilter = rating ? { city } : {}  const countTours = await Tour.countDocuments({ ...categoryFilter, ...countryFilter, ...cityFilter })  const tours = await Tour.find({  ...categoryFilter,  ...countryFilter,  ...cityFilter,  rating: {$gt : rating},  price: {$gt : priceFrom, $lt: priceTo}  }).skip(pageSize \* (currentPage - 1)).limit(pageSize).populate('categoryS')  res.send({ tours, pages: Math.ceil(countTours / pageSize) })  })  tourRouter.get('/home', async (req, res) => {  /\* const category2 = await Category.create({ categoryName: 'History', categoryDesc : 'Historical travel includes various kinds of destinations, from Stone Age cave paintings to Cold War installations of the late 20th century.'})  await category2.save() \*/  const category = req.query.category  const country = req.query.country  const city = req.query.city  const categoryFilter = category ? { category } : {}  const countryFilter = country ? { country } : {}  const cityFilter = city ? { city } : {}  const tours = await Tour.find({ ...categoryFilter, ...countryFilter, ...cityFilter}).select('\_id title image category label desc additionalInfo price rating numReviews country city').sort({rating: -1})  res.send({tours})  })  tourRouter.get('/categories', async (req, res) => {  const categories = await Tour.find().distinct('category')  res.send(categories)  })  tourRouter.get('/country', async (req, res) => {  const country = await Tour.find().distinct('country', )  res.send(country)  })  tourRouter.get('/cityInTheCountry2', async (req, res) => {  const country = req.query.country  if (country) {  const countryList = await Tour.find().distinct('city', { 'country' : country })  res.send(countryList)  } else {  res.send({ message: 'Set the country in the request parameter'})  }  })  tourRouter.get('/city', async (req, res) => {  const cities = await Tour.find().distinct('city')  res.send(cities)  })  tourRouter.get('/getHistory', async (req, res) => {  const userId = req.query.userId  const tourHistory = await User.findById(userId).sort({'visitShema.counter' : 1}).populate('visitShema.tourId')  const tourHistory2 =await User.aggregate([  { $match: {\_id: ObjectId(userId)}},  { $unwind: "$visitShema" },  { $sort: { "visitShema.updatedAt": -1 } },  {  $lookup:  {  from: "tours",  localField: "visitShema.tourId",  foreignField: "\_id",  as: "visitShema.tourId"  }  }  ]);  res.send(tourHistory2)  })  tourRouter.get('/getRecomendation', async (req, res) => {  const recomedation = await User.aggregate([  { $match: {\_id: ObjectId(req.query.userId)}},  { $unwind: "$visitShema" },  { $project: {  visitShema : 1  }},  { $sort: {'visitShema.counter': -1} },  { $lookup :  {  from: "tours",  localField: "visitShema.tourId",  foreignField: "\_id",  pipeline : [{  $project: {  "title": 1,  "price" : 1,  "rating" : 1,  "category" : 1,  "country" : 1,  "city" : 1,  "rating" : 1  }  }],  as: "visitShema.tourId"  },  },  {$group:  {  \_id: "$visitShema.tourId.city",  maxcounter: { "$max": "$visitShema.counter"}  }  },  ])  const bestTourInMostVisitCountry = []  if (recomedation.length >= 1) {  const bestTourInMostVisitCountry1 = await Tour.find({city : recomedation[0].\_id}).sort({rating : -1}).select('\_id title image category label desc additionalInfo price rating numReviews country city').limit(1)  bestTourInMostVisitCountry.push(bestTourInMostVisitCountry1[0])  }  if (recomedation.length >= 2) {  const bestTourInMostVisitCountry2 = await Tour.find({city : recomedation[1].\_id}).sort({rating : -1}).select('\_id title image category label desc additionalInfo price rating numReviews country city').limit(1)  bestTourInMostVisitCountry.push(bestTourInMostVisitCountry2[0])  }  if (recomedation.length >= 3) {  const bestTourInMostVisitCountry3 = await Tour.find({city : recomedation[2].\_id}).sort({rating : -1}).select('\_id title image category label desc additionalInfo price rating numReviews country city').limit(1)  bestTourInMostVisitCountry.push(bestTourInMostVisitCountry3[0])  }  let mostVisitTours = await User.aggregate([  { $match: {\_id: ObjectId(req.query.userId)}},  { $unwind: "$visitShema" },  { $project: {  visitShema : 1  }},  { $sort: {'visitShema.counter': -1} },  { $lookup :  {  from: "tours",  localField: "visitShema.tourId",  foreignField: "\_id",  pipeline : [{  $project: {  "title": 1,  "price" : 1,  "rating" : 1,  "category" : 1,  "country" : 1,  "city" : 1,  "rating" : 1,  "image" : 1  }  }],  as: "visitShema.tourId"  },  },  ])  res.send({  mostVisitTours,  bestTourInMostVisitCountry  })  })  tourRouter.get('/:id', async (req, res) => {  let tourId = req.params.id  let userId = req.query.userId  if (req.query.userId) {  const user = await User.find({ "\_id" : userId, "visitShema.tourId" : tourId})  if (user.length) {  await User.updateOne(  { \_id: userId, "visitShema.tourId": tourId },  { $inc: { "visitShema.$.counter": 1 } ,  $set: { "visitShema.$.updatedAt" : Date.now() }  },  )  } else {  const visit = { tourId, counter : 1, updatedAt: Date.now() }  const user = await User.findById(userId)  user.visitShema.push(visit)  const updatedUser = await user.save()  }    /\* let averageRatingTour = await User.aggregate(  [  { $match: {\_id: userId}},  {  $sort: { "visitShema.$.updatedAt": 1 }  }  ]  )  console.log(averageRatingTour) \*/  }  const tour = await Tour.findById(req.params.id).populate('comments.user').populate('categoryS cityT')  if (tour) {  res.send(tour)  } else {  res.status(404).send({message: 'Tour not Found'})  }  })  tourRouter.get('/seed', async (req, res) => {  const createdTours = await Tour.insertMany(data.tours)  res.send({createdTours})  })  tourRouter.post('/', isAuth, isAdmin, async (req, res) => {  const tour = new Tour({  title: 'title',  image: '/images/img-2.jpg',  category: "category",  label :"label",  desc:"desc",  additionalInfo: "additionalInfo",  price: 0,  rating: 0,  numReviews: 0,  country: 'country'  })  const createdTour = await tour.save()  res.send({ message: 'Tour created', tour: createdTour})  })  tourRouter.put('/:id', isAuth, isAdmin, async (req, res) => {  const categoryFrom = await Category.findOne({categoryName: req.body.category})  const city = await City.findOne({cityName: req.body.city})  const tourId = req.params.id  const tour = await Tour.findById(tourId)  if (tour) {  tour.title = req.body.title;  tour.image = req.body.image;  tour.category = req.body.category;  tour.label = req.body.label;  tour.desc = req.body.desc;  tour.additionalInfo = req.body.additionalInfo;  tour.price = req.body.price;  tour.country = req.body.country;  tour.city = req.body.city;  tour.cityT = city.\_id;  tour.imageGallery = req.body.uploadedImage.length !== 0 ? req.body.uploadedImage : tour.imageGallery,  tour.categoryS = categoryFrom.\_id  const updatedTour = await tour.save()  res.send({ message: 'Tour updated', tour: updatedTour})  } else {  res.status(404).send({ message: 'Tour not found' })  }  })  tourRouter.delete('/:id', isAuth, isAdmin, async (req, res) => {  const tour = await Tour.findById(req.params.id)  if (tour) {  const deletedTour = await tour.remove()  res.send({ message: 'Tour deleted', tour: deletedTour })  } else {  res.status(404).send({ message: 'Tour not found'})  }  })  tourRouter.get('/search/:title', async (req, res) => {  const title = req.params.title  const titleFilter = title ? { title: { $regex: title, $options: 'i' } } : {};  const tours = await Tour.find({...titleFilter})  res.send(tours)  })  tourRouter.post('/:id/comments', isAuth, async (req, res) => {  const tourId = req.params.id  const tour = await Tour.findById(tourId)  if (tour) {  const checkOrder = await Order.find({ userInfo : req.user.\_id, orderItems: tourId } )  if (checkOrder.length !== 0 || req.user.isAdmin) {  const comment = {  user: req.user.\_id,  comment: req.body.comment,  rating: Number(req.body.rating)  }  tour.comments.push(comment)  tour.numReviews = tour.comments.length  tour.rating = (tour.comments.reduce((a, c) => c.rating + a, 0 ) / tour.comments.length).toFixed(2)  const updatedTour = await tour.save()  res.status(201).send({ message: 'Комментарий добавлен', comment: updatedTour.comments[updatedTour.comments.length - 1]})  } else {  res.status(404).send({ message: 'Вы не можете писать комментарий, так как вы не заказывали данный тур.' })  }  } else {  res.status(404).send({ message: 'Tour not found' })  }  })  tourRouter.get('/:id/comments', async (req, res) => {  const tourId = req.params.id  const tour = await Tour.findById(tourId).populate('comments.user')  if (tour) {  res.status(201).send(tour.comments)  } else {  res.status(404).send({ message: 'Tour not found' })  }  })  module.exports = tourRouter |

ПРИЛОЖЕНИЕ К

Листинг роутера панели администратора

|  |
| --- |
| const express = require('express')  const Order = require('../models/orderModel')  const Tour = require('../models/tourModel')  const Country = require('../models/countryModel')  const City = require('../models/cityModel')  const User = require('../models/userModel')  const { isAuth, isAdmin } = require('../middleware/utils')  var mongoose = require('mongoose');  const ObjectId = require('mongodb').ObjectID;  const Category = require('../models/categoryModel')  const dashboardRouter = express.Router()  dashboardRouter.get('/summary', async (req, res) => {  const countOrders = await Order.find().count() //+  const countOrdersTours = await Order.find()  let arrOrders = [], arrPaidOrders = []  countOrdersTours.forEach(el => el.orderItems.forEach(el2 => arrOrders.push(el2)))  countOrdersTours.forEach(el => { if (el.isPaid) arrPaidOrders.push(el) })  const totalCostByCategory = await Tour.aggregate(  [  {$match: {}},  {  $group: {  \_id: "$categoryS",  total: {$sum: "$price"}  }  },  {  $lookup:  {  from: "сategories",  localField: "\_id",  foreignField: "\_id",  as: "categoryName"  }  }  ]  )  const tourCount = await Tour.find().count()  const dailyOrders = await Order.aggregate([  {  $group: {  \_id: { $dateToString: { format: '%Y.%m.%d', date: '$createdAt' } },  orders: { $sum: 1 },  sales: { $sum: '$totalPrice' },  },  },  { $sort: { \_id: 1 } },  ]);  const averagePriceOnTour = await Tour.aggregate([  {  $group: {  \_id:"$country",  avg\_price: {  $avg: "$price"  }  } ,  },  {  $sort: {'total\_st': -1}  }  ])  let orderInDayAverage = 0  dailyOrders.forEach(el => orderInDayAverage += el.orders)  let date1 = new Date(dailyOrders[dailyOrders.length - 1].\_id)  let date2 = new Date(dailyOrders[0].\_id)  let result = ( date1 - date2 ) / 864e5  let orderInDay = (orderInDayAverage / result).toFixed(2)  const findPopularCategory = await Order.aggregate([  { $unwind: "$orderItems" },  { $group: {  \_id: "$orderItems",  count: { "$sum": 1 }  }},  { "$sort": { "count": -1 } },  {  $lookup:  {  from: "tours",  localField: "\_id",  foreignField: "\_id",  as: "tour"  },    }  ])  const countyCount = await Country.find().count()  const cityCount = await City.find().count()  let averageRatingTour = await Tour.aggregate(  [{  $group: {  \_id: null,  avgRating: { $avg: '$rating' }  }  }]  )  averageRatingTour = (averageRatingTour[0].avgRating).toFixed(2)  let averagePriceTour = await Tour.aggregate(  [  {  $group: {  \_id: null,  avgPrice: { $avg: '$price' }  }  }  ])  averagePriceTour = (averagePriceTour[0].avgPrice).toFixed(2)  const arrayOfToursVisits = await User.aggregate([  { $project : { "visitShema" : 1 } },  { $unwind: "$visitShema" },  {$group: {  \_id: "$visitShema.tourId",  count: { "$sum": "$visitShema.counter" }  } },  {  $lookup:  {  from: "tours",  localField: "\_id",  foreignField: "\_id",  pipeline : [{  $project: {  "title": 1,  }  }],  as: "\_id"  },},  { $sort: { count: -1 } },  ]);  console.log(arrayOfToursVisits)  res.send({  countOrders: countOrders,  countOrdersTours : arrOrders.length,  paidOrders : arrPaidOrders.length,  totalCostByCategory,  tourCount,  orderInDay,  dailyOrders,  averagePriceOnTour,  findPopularCategory,  countyCount,  cityCount,  averageRatingTour,  averagePriceTour,  arrayOfToursVisits  })  })  module.exports = dashboardRouter |

ПРИЛОЖЕНИЕ Л

Листинг *React*-компонента *Tour*

|  |
| --- |
| import React, { useState, useEffect } from "react";  import { useDispatch, useSelector } from 'react-redux'  import { detailsTour, commentCreate, commentDelete } from "../redux/actions/tourActions";  import DatePicker from "react-datepicker";  import LoadingBox from "../components/LoadingBox";  import MessageBox from "../components/MessageBox";  import "react-datepicker/dist/react-datepicker.css";  import '../components/MainSection.css';  import { TOUR\_COMMENT\_CREATE\_RESET } from '../redux/constants/tourConstants'  import { Link } from 'react-router-dom';  import Axios from "axios"  import Rating from '../components/rating'  import { commentUpdateStatus } from '../redux/actions/tourActions'  import { YMaps, Map, Clusterer, Placemark, FullscreenControl, GeolocationControl, TypeSelector, ZoomControl, ListBox, ListBoxItem } from 'react-yandex-maps';  import '../../node\_modules/react-image-gallery/styles/css/image-gallery.css'  import ImageGallery from 'react-image-gallery'  import env from "dotenv"  env.config()  function Tour(props) {  const dispatch = useDispatch()  const tourId = props.match.params.id  const [startDate, setStartDate] = useState(new Date());  const [endDate, setEndDate] = useState(new Date(new Date().setDate(new Date().getDate() + 7)));  const [rating, setRating] = useState(0);  const [emptyError, setEmptyError] = useState(false);  const [emptyError2, setEmptyError2] = useState(false);  const [updateRating, setUpdateRating] = useState(0);  const [comment, setComment] = useState('');  const [comments, setComments] = useState('');  const [commentsCreated, setCommentsCreated] = useState(false);  const [deletedComment, setDeletedComment] = useState([])  const [arrayImageTour, setArrayImageTour] = useState([])  const [centerMap, setCenterMap] = useState({})  let [numComments, setNumComments] = useState('')  const tourDetails = useSelector(state => state.tourDetails)  const { loading, error, tour } = tourDetails  const userSignIn = useSelector(state => state.userSignIn)  const { userInfo } = userSignIn  const tourCommentCreate = useSelector((state) => state.tourCommentCreate);  const { loading: loadingCommentCreate, error: errorCommentCreate, success: successCommentCreate, } = tourCommentCreate;  const onChange = (dates) => {  const [start, end] = dates;  setStartDate(start);  setEndDate(end);  };  const onItemClick = coords => {  setCenterMap({ center: coords });  };  const getComments = async (tourId) => {  const tourInfo = await Axios.get(`/api/tours/${tourId}`)  setComments(tourInfo.data.comments.reverse())  setUpdateRating(tourInfo.data.rating)  setNumComments(tourInfo.data.numReviews)  /\* setComments(data.data.reverse()) \*/  }  const addToCartHandler = () => {  props.history.push(`/cart/${tourId}?startDate=${startDate}&endDate=${endDate}&count=${count}`)  }  const submitHandler = (e) => {  e.preventDefault();  if (comment && rating) {  if (comment.length < 1000) {  dispatch(  commentCreate(tourId, { comment, rating, user: userInfo.\_id })  );  } else {  setEmptyError2(true)    }  setEmptyError(false)  } else {  setEmptyError2(false)  setEmptyError(true)  }  };  const submitUpdateStatusComment = async (commentId, status) => {  await dispatch(commentUpdateStatus(commentId, tourId, status))  getComments(tourId)  }  const submitDeleteComment = (commentId) => {  dispatch(commentDelete(commentId, tourId))  setDeletedComment(arr => [...deletedComment, commentId])  let countActiveComments = numComments - 1  setNumComments(countActiveComments)  }  useEffect(() => {  let newA = []  if (tour) {  tour.imageGallery.forEach(e => { let obj = { original: e, thumbnail: e }; newA.push(obj) })  setArrayImageTour(newA)  let objState = {  center: [tour.cityT.lat, tour.cityT.lon],  zoom: 10,  behaviors: ['default', 'scrollZoom']  }  setCenterMap(objState)  }  }, [dispatch, tour])  useEffect(() => {  console.log('111111', tourId)  dispatch(detailsTour(tourId))  getComments(tourId)  }, [dispatch, tourId])  useEffect(() => {  if (successCommentCreate) {  setRating('');  setComment('');  dispatch({ type: TOUR\_COMMENT\_CREATE\_RESET });  getComments(tourId)  setCommentsCreated(true)  }  }, [dispatch, successCommentCreate, tourId])  const [count, setCount] = useState(2)  const decrement = () => {  if (count > 1) setCount(count - 1)  }  const increment = () => {  if (count < 12) setCount(count + 1)  }  const getPointData = (index, title, desc, image) => {  return {  balloonContentBody: [  `<i>Название: ${title}</i>`,  '<br/><br/>',  `<strong>${desc}</strong>`,  '<br/>',  `<img alt="" className="grid\_\_image" src=${image} />`,  '<br/>',  ].join(''),  clusterCaption: 'placemark <strong>' + index + '</strong>',  };  };  const getPointOptions = () => {  return {  preset: 'islands#blackIcon',  };  };  return (  <>  {loading ? (<div className="loading"><LoadingBox></LoadingBox></div>)  :  error ? (<div class="message\_box\_tour"><MessageBox variant="danger">{error}</MessageBox></div>)  : (  <>  <div className='grid'>  <div className='grid\_\_section'>  <div className='grid\_\_content\_\_container'>  <div className='grid\_\_content'>  <h1 className='text-center'>  {tour.title}  </h1>  <br /> <br />  <div className="text-divider\_\_divider"></div>  <br />  <h3 className='text-center2'>  {tour.desc}  </h3>  <br />  <div className="text-divider\_\_divider"></div>  <br />  <h2 className='text-center2'>  <Rating rating={updateRating && updateRating} numReviews={updateRating && updateRating} />  </h2>  </div>  </div>  </div>  <div className='grid\_\_section '>  <div className='grid\_\_wrap'>  <button className='grid\_\_gallery-button' >  <img alt="" className="grid\_\_image" src={tour.image} />  </button>  </div>  </div>  </div>  <section className="main-section">  <div className="main-container">  <div className="grid-section">  <div className="grid-sidebar">  <div className='head-text'>  Добавить тур в корзину  </div>  <div className="grid-with-sidebar">  <DatePicker  selected={startDate}  onChange={onChange}  startDate={startDate}  endDate={endDate}  selectsRange  inline  />  <div className="box-head">Количество человек</div>  <div className="counterCartTour">  <button onClick={decrement} className="form-incrementer\_\_btn" data-decrement="" type="button">  <svg xmlns="http://www.w3.org/2000/svg" width="26" height="26" viewBox="0 0 26 26" fill="#1a1a1a">  <g transform="translate(3.603 3)">  <path d="M5.3,9h8.3v2H5.3V9z"></path>  </g>  <path  d="M13,2c6.1,0,11,4.9,11,11s-4.9,11-11,11S2,19.1,2,13S6.9,2,13,2 M13,1C6.4,1,1,6.4,1,13s5.4,12,12,12s12-5.4,12-12 S19.6,1,13,1L13,1z">  </path>  </svg>  </button>  <input type="number" id="booking-pax" value={count}  className="form-incrementer\_\_input form-incrementer\_\_input\_\_booking" required="required"  readOnly="readonly" />  <button onClick={increment} className="form-incrementer\_\_btn" data-increment="" type="button">  <svg xmlns="http://www.w3.org/2000/svg" width="26px" height="26px" viewBox="0 0 26 26" fill="#1a1a1a">  <path  d="M13,2c6.1,0,11,4.9,11,11s-4.9,11-11,11S2,19.1,2,13S6.9,2,13,2 M13,1C6.4,1,1,6.4,1,13s5.4,12,12,12s12-5.4,12-12 S19.6,1,13,1L13,1z">  </path>  <g transform="translate(4 3)">  <path className="st1" d="M8,9V4.8H10V9h4.2V11H10v4.2H8V11H3.8V9H8z"></path>  </g>  </svg>  </button>  </div>  <input onClick={addToCartHandler} type="submit" value="Добавить в корзину" class="action-button-tour" />  </div>  </div>  <div>  </div>  <div className="grid-main-column">  <div className='head-text'>  Tour Details  </div>  <div className="box">  <div className="box-head">  Tour description  </div>  <div className="box-body">  {tour.desc}  {tour.additionalInfo}  </div>  <div className="box-head">  Tour category - {tour.categoryS.categoryName}  </div>  <YMaps query={{ apikey: process.env.REACT\_APP\_API\_KEY\_YANDEX\_MAPS }}>  <Map width='100%' height='500px' state={centerMap && centerMap}>  <Clusterer  options={{  preset: 'islands#invertedVioletClusterIcons',  groupByCoordinates: false,  clusterDisableClickZoom: true,  clusterHideIconOnBalloonOpen: false,  geoObjectHideIconOnBalloonOpen: false,  }}  >  {tour.attractions.map((a, idx) =>  <Placemark  geometry={[a.lat, a.lon]}  key={idx}  options={getPointOptions()}  properties={getPointData(idx, a.titleAttraction, a.descAttraction, a.imageAttraction)}  modules={['geoObject.addon.balloon', 'geoObject.addon.hint']}  />)  }  </Clusterer>  <FullscreenControl />  <GeolocationControl options={{ float: 'left' }} />  <TypeSelector options={{ float: 'right' }} />  <ZoomControl options={{ float: 'right' }} />  </Map>  </YMaps>    <ImageGallery items={arrayImageTour} />  </div>  <div className='head-text'>  Добавьте комментарий  </div>  <div className="box">  <div>  <ul>  <li>  {userInfo ? (  <form className="form" onSubmit={submitHandler}>  <div className="box-head">  Напишите комментарий  </div>    {emptyError && <MessageBox variant="danger">Пожалуйста напишите комментарий и поставьте оценку туру.</MessageBox>}  {emptyError2 && <MessageBox variant="danger">Длина комментария должна быть не более 1000 символов.</MessageBox>}  {commentsCreated && (  <MessageBox variant="success">  Комментарий успешно добавлен  </MessageBox>  )}  {errorCommentCreate && (  <MessageBox variant="danger">  {errorCommentCreate}  </MessageBox>  )}  <div>  <label htmlFor="rating">Оценка</label>  <select  id="rating"  value={rating}  onChange={(e) => setRating(e.target.value)}  >  <option value="">Выберите...</option>  <option value="1">1- Бедно</option>  <option value="2">2- Плохо</option>  <option value="3">3- Нормально</option>  <option value="4">4- Очень хорошо</option>  <option value="5">5- Великолепно</option>  </select>  </div>  <div>  <label htmlFor="comment">Комментарий</label>  <textarea  id="comment"  value={comment}  onChange={(e) => setComment(e.target.value)}  ></textarea>  </div>  <div>  <button className="action-button-tour" type="submit">  Отправить  </button>  </div>  <div>  {loadingCommentCreate && <LoadingBox></LoadingBox>}  </div>  </form>  ) : (  <MessageBox>  Please <Link to="/signUp">Sign In</Link> to write a comment  </MessageBox>  )}  </li>  </ul>  </div>  </div>  <div className='head-text'>  Комментарии  </div>  <div className="box">  <h2 className="box-head">Комментарии {numComments && ' - total ' + numComments}</h2>  {comments.length === 0 && (  <MessageBox>There is no comment</MessageBox>  )}  <ul>  {comments && comments.filter(row => !deletedComment.includes(row.\_id) && (row.isActive)).map(comment => (  /\* {comments && comments.map((comment) => ( \*/  <li key={comment.\_id}>  <div className="card\_\_name">{comment.user.firstName} {comment.user.lastName}  </div>  <p>Comment: {comment.comment}</p>  {/\* <Rating rating={review.rating} caption=" "></Rating> \*/}  <p>Status: {comment.isActive.toString()}</p>  <p>{comment.createdAt.substring(0, 10)}  <Rating rating={comment.rating} numReviews={comment.rating} />  </p>  {userInfo && userInfo.isAdmin &&  <div className="btn-remove-comment">  <button className="btn-remove" type="button" onClick={(e) => submitUpdateStatusComment(comment.\_id, 'disable')}>Отключить</button>  {/\* <button className="btn-remove" type="button" onClick={(e) => submitUpdateStatusComment(comment.\_id, 'enable')}>enable</button> \*/}  <button className="btn-remove" type="button" onClick={(e) => submitDeleteComment(comment.\_id)}>Удалить</button>  </div>  }  <div className="text-divider\_\_divider"></div>  <br />  </li>  ))}  </ul>  </div>  {userInfo && userInfo.isAdmin && <>  <div className='head-text'>  Отключенные комментарии  </div>  <div className="box">  <h2 className="box-head">Disabled comments </h2>  {comments.length === 0 && (  <MessageBox>Нет комментарий</MessageBox>  )}  <ul>  {comments && comments.filter(row => !deletedComment.includes(row.\_id) && (!row.isActive)).map(comment => (  /\* {comments && comments.map((comment) => ( \*/  <li key={comment.\_id}>  <div className="card\_\_name">{comment.user.firstName} {comment.user.lastName}  </div>  <p>Comment: {comment.comment}</p>  {/\* <Rating rating={review.rating} caption=" "></Rating> \*/}  <p>Status: {comment.isActive.toString()}</p>  <p>{comment.createdAt.substring(0, 10)}  <Rating rating={comment.rating} numReviews={comment.rating} />  </p>  {userInfo && userInfo.isAdmin &&  <div className="btn-remove-comment">  {/\* <button className="btn-remove" type="button" onClick={(e) => submitUpdateStatusComment(comment.\_id, 'disable')}>disable</button> \*/}  <button className="btn-remove" type="button" onClick={(e) => submitUpdateStatusComment(comment.\_id, 'enable')}>Одобрить</button>  <button className="btn-remove" type="button" onClick={(e) => submitDeleteComment(comment.\_id)}>Удалить</button>  </div>  }  <div className="text-divider\_\_divider"></div>  <br />  </li>  ))}  </ul>  </div>  </>}  </div>  </div>  </div>  </section>  </>  )  }  </>  )  }  export default Tour |