**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Расчетно-графическое задание

по дисциплине: Технологии Web-программирования

тема: «**Веб-приложение «Путеводитель по Исландии»**»

Выполнил: ст. группы ПВ-41

Таран Алексей Николаевич

Проверил: Картамышев С.В.

Белгород 2020

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc58682614)

[Разработка макетов и верстка 3](#_Toc58682615)

[Содержание страниц приложения 3](#_Toc58682616)

[Создание макетов страниц средствами HTML 4](#_Toc58682617)

[Стилизация страниц при помощи CSS и Bootstrap 6](#_Toc58682618)

[Клиентское программирование 8](#_Toc58682619)

[Фреймворк Vue JS 8](#_Toc58682620)

[Добавление необходимых компонентов и перенос верстки 8](#_Toc58682621)

[Серверное программирование 9](#_Toc58682622)

[Настройка базового приложения PHP YII2 9](#_Toc58682623)

[Настройка конфигурации приложения для работы с Docker 10](#_Toc58682624)

[Добавление модуля с API 11](#_Toc58682625)

[Проектирование базы данных 12](#_Toc58682626)

[Выбор СУБД и базовая конфигурация 12](#_Toc58682627)

[Разработка структуры 14](#_Toc58682628)

[Разработка моделей 15](#_Toc58682629)

[Разработка REST API 20](#_Toc58682630)

[Особенности разработки REST API при помощи YII2 20](#_Toc58682631)

[Документация к API 20](#_Toc58682632)

[Клиентские HTTP запросы 28](#_Toc58682633)

[Axios JS - средство для отправки AJAX запросов на сервер 28](#_Toc58682634)

[Взаимодействие клиентской и серверной части приложения 28](#_Toc58682635)

[Скриншоты 29](#_Toc58682636)

[Заключение 31](#_Toc58682637)

[Список литературы 32](#_Toc58682638)

**ВВЕДЕНИЕ**

В данной работе мы поэтапно разработаем приложение «Путеводитель по Исландии». Вначале, опишем будущие страницы, а затем сверстаем их. Выберем клиентский фреймворк для отображения страниц. Развернем серверную часть приложения. Напишем базу данных и оформим в виде REST API на стороне сервера запросы к этой базе. Соединим клиентскую и серверную часть путем совершения HTTP запросов по написанному API.

**Разработка макетов и верстка**

**Содержание страниц приложения**

Приложение «Путеводитель по Исландии» должно содержать следующие страницы:

1) Обзор

2) Страница города

3) Что посмотреть

4) Кухня

5) Как добраться

6) Где жить

7) Войти

Каждая страница должна содержать header («шапка») и footer («нижняя часть»).

Шапка содержит ссылки на страницы городов, достопримечательностей, кухни, как добраться и где жить. Также позволяет пользователю зарегистрироваться или войти в систему.

Нижняя часть содержит общую информацию о правах и создателях сайта.

Остановимся подробнее на каждой странице:

Обзорная страница содержит базовую информацию о стране, несколько галерей фотографий, видео о стране, заметки с лайфхаками. Меню разделов содержит ссылки на столицу страны – город Рейкъявик, страницу достопримечательностей, кухни, способа добраться, где жить и на заметки.

Страница города содержит разделы с информацией о городе. Например, она описывает историю, географию, климат, интересное, покупки, транспорт, раздел «Рекомендуем». Внутри каждого раздела поместить фотографии. А также добавить фотографии на страницу в целом.

Страница достопримечательностей помимо разделов с достопримечательностями содержит в нижней части раздел комментариев. Все комментарии – могут просматривать все пользователи, но редактировать и удалять определенные могут лишь их создавшие пользователи.

Страницы кухни, способов передвижения, способов проживания аналогичны, отличаясь лишь содержанием контента. Они содержат подразделы, более точечно описывая одну из составляющих предметной области.

Окно входа в систему/регистрации содержит формы ввода логина и пароля. После успешного входа/выхода из системы информация о пользователе меняется на header панели. Также, состояние входа в систему влияет на права по отношению к комментариям.

**Создание макетов страниц средствами HTML**

Макеты страниц создаем при помощи HTML-разметки.

Основные HTML-теги:

<html></html> - корневой тег, который сообщает браузеру, что это HTML-документ. Все остальные элементы помещаются внутри него.

<head></head> - контейнер, в который помещаются метаданные документа, не видимые пользователям, но считываемые поисковыми роботами.  
 <meta> - тег для оформления метаданных документа, используемых браузером для обработки страницы, а поисковиками — для индексации.

<body></body> - тег, обрамляющий видимую пользователям часть документа. Всё, что вы укажите внутри этого контейнера, отобразится на странице. Тег <body> имеет несколько атрибутов, позволяющих управлять цветами.

Для форматирования текста:

<h1></h1>......<h6></h6>. — теги заголовков, от самого большого к самому маленькому.

<b></b> — жирный текст без придания важности выделенному фрагменту.

<strong></strong> — расставление акцентов в тексте путём выделения его фрагментов полужирным начертанием.

<i></i> — выделение текста курсивом без придания важности.

<p></p> — контейнер для абзаца.

Встраиваемые элементы:

<img></img> — вставка изображения. Имеет атрибут src, который указывает на адрес нужного файла. Есть и другие атрибуты:

<img src="name" align=?> — выравнивание к одной из сторон документа. Например, значение right переместит рисунок в правый край, а left — в левый.

<img src="name" border=?> — позволяет настроить в пикселях толщину рамки вокруг изображения.  
  
 <picture></picture> — контейнер для расположения элемента <img> без <source>, который даёт браузеру возможность самостоятельно выбрать подходящую картинку.

<audio></audio>  — вставка звукового контента.

<video></video> — вставка видео (поддерживается Ogg, WebM и MP4).

**Стилизация страниц при помощи CSS и Bootstrap**

Настройка стилей HTML-документов осуществляется при помощи CSS. Подключение css в HTML-файл осуществляется в head при помощи следующей команды: *<link rel="stylesheet" href="../css/homePage.css">*.

Типичная структура CSS файла аналогична словарю содержащим пары ключ-значение. Приведем пример создание стиля:

*.cssStyle{*

*color:rgba(0, 0, 0, 0.9);*

*font-family: 'PT Serif';*

*font: bold;*

*letter-spacing: 0px;*

*text-decoration: none;*

*}*

Подключение стиля осуществляется в HTML-файле в class при помощи *class="cssStyle"*

Внешний вид страниц должен выглядеть приятным с точки зрения пользователя, поэтому воспользуемся сторонней библиотекой Bootstrap, которая предоставляет удобные компоненты с точки зрения графического содержания и автоматической оптимизации отображения под мобильные устройства с малым экраном.

Подключение библиотеки, а также используемых JS скриптов осуществляется аналогично CSS файлам – в head документа.

Основные использованные bootstrap компоненты navbar, сard, modal, btn, carousel, footer-copyright и другие.

Для компонентов проводилась настройка css (например, используемых шрифтов).

Приведем примеры некоторых разработанных частей страниц:

A picture containing text, different, several

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Клиентское программирование**

**Фреймворк Vue JS**

Это веб-фреймворк, который может быть использован для создания одностраничных проектов в реактивном стиле на JavaScript. Включает в себя: vue-cli – инструментарий для разработки на vue js, vue-router – маршрутизатор для vue, а также vue-loader – загрузчик веб-пакетов.

Vue-cli позволяет создать новый проект при помощи *vue create newProject.* Далее, выбираются необходимые параметры с точки зрения пользователя.

**Добавление необходимых компонентов и перенос верстки**

В файле index.js есть массив routes, который необходимо заполнить в соответствии с таблицей маршрутизации проекта. Например, для страницы достопримечательности соответствующий элемент выглядит следующим образом:

*{*

*path: '/attractions',*

*component: () => import('@/views/attractionsPage.vue'),*

*meta: { title: 'Достопримечательности' }*

*},*

Основой фреймворка являются компоненты, позволяющие использовать их повторно для различных страниц и различного контента. Также, из-за одностраничности приложения следует то, что при загрузке нового Vue компонента (который может представлять собой страницу), если явно не указать перезагрузку страницы, то, по умолчанию, старое содержимое заменится новым без перезагрузки.

Синтаксис описания компонентов следующий:

Внутри тега <template></template>, включается компонент:

*<listChapters*

*v-bind:listChaptersPlaceholder="listChapters"*

*v-bind:generalNameChaptersPlaceholder="generalNameChapters"*

*v-bind:generalLogoChaptersPlaceholder="generalLogoChapters"*

*v-bind:getCountStringsFunction="getStringIndexesForChapters"*

*v-bind:setScreenNameFunction="setPageName"*

*/>*

Импорт компонентов осуществляется внутри тега <sсript> при помощи команды: import component from «component.vue». Используемые компоненты описываются в export default{сomponents: …}.

В export default{props: …} описываются компоненты, принятые от родительского компонента.

Данные, появившиеся впервые в компоненте описываются в export default{сomponents: …}.

v-bind – означает соответствие названия функции в компоненте-приемнике (до равно) и функции в компоненте, передающим название (после равно). Функции описываются в export default{data(){return {data1: …, data2:…}} …}.

При помощи v-on/@ можно реализовать реакцию на нажатие чего-либо на дочернем компоненте, при помощи отправки сообщения обратно родителю при помощи emit.

v-for позволяет совершить итерацию по элементам коллекции.

Подключение css файлов осуществляется в теге <style>:

*<style lang="css">*

*@import '../components/projectStyle.css'*

*</style>*

Перенос верстки и настройка маршрутизации осуществляется в соответствии с правилами, указанными выше.

Дополнительно используем то, что для загрузки картинок на странице, их нужно предварительно обернуть в require для нужд webpack.

**Серверное программирование**

**Настройка базового приложения PHP YII2**

Серверную часть приложения напишем при помощи фреймворка Yii2.

При помощи команды *curl -sS https://getcomposer.org/installer | php*

*sudo mv composer.phar /usr/local/bin/composer* установим composer, а также фреймворк на машину.

Затем при помощи команды composer create-project --prefer-dist yiisoft/yii2-app-basic newapp создаем новое yii2 приложение.

Результат работы: новые созданные папки и файлы.

Table

Description automatically generated

**Настройка конфигурации приложения для работы с Docker**

Для изоляции окружения приложения серверной части, при помощи docker-compose соберем контейнеры из серверного приложения yii2 и будущей базы данных mysql. Напишем соответствие папки в контейнере с папкой с приложением на хосте, а также укажем соответствие портов на хосте и контейнере. Применим указанные параметры для написания docker-compose.yml файла. Запуск docker-контейнеров осущеcтвляется при помощи docker-compose up -d (в режиме демона).

*version: '3.1'*

*services:*

*php:*

*image: yiisoftware/yii2-php:7.3-apache*

*volumes:*

*- ~/.composer-docker/cache:/root/.composer/cache:delegated*

*- ./:/app:delegated*

*ports:*

*- '1199:80'*

*extra\_hosts:*

*- "visiticelandbeauty.com:127.0.0.1"*

*mysql:*

*image: mysql*

*command: --default-authentication-plugin=mysql\_native\_password --character-set-server=utf8 --collation-server=utf8\_general\_ci*

*environment:*

*- MYSQL\_ROOT\_PASSWORD=verysecret*

*- MYSQL\_DATABASE=bstu*

*- MYSQL\_USER=bstu*

*- MYSQL\_PASSWORD=bstu321!*

**Добавление модуля с API**

Добавляем поддержку API в файле web/config, где разрешаем обращаться к API при помощи 127.0.0.1:port/api/v1…. Приведем этот фрагмент:

*'db' => $db,*

*'urlManager' => [*

*'enablePrettyUrl' => true,*

*'showScriptName' => false,*

*'rules' => [*

*'api/<module:\w+>/<controller:\w+>/<action:\w+>' => '<module>/<controller>/<action>'*

*],*

*],*

*'params' => $params,*

*'modules' => [*

*'v1' => [*

*'class' => 'app\modules\v1\v1'*

*]*

*]*

*];*

Создадим ApiController, который содержит базовые настройки, присущие будущим контроллерам REST API: CompositeAuth, ContentNegotiator, Cors, RateLimiter, VerbFilter, Controller, Response.

Наследуясь от ApiController, создаем для тестирования класс, присущий контроллеру главной страницы. В качестве ее методов представлены реакции на API-запросы клиента. Пример:

/\*\*

\* Пример запроса:

\* http://127.0.0.1:1199/api/v1/mainpage/names

\*/

public function actionNames(){

$mainPageObject = new MainPage();

return $mainPageObject -> dataTable();

}

Класс MainPage еще не описан. Это имитация данных, т.е представляет собой модель. Такой класс должен быть наследован относительно *use yii\base\Model;* Запишем возврат фиктивных данных по вызову dataTable() и протестируем работу API в POSTMAN

Graphical user interface, application

Description automatically generated

**Проектирование базы данных**

**Выбор СУБД и базовая конфигурация**

В качестве базы данных выберем MySQL, которая была выбрана при создании docker-контейнера:

*mysql:*

*image: mysql*

*command: --default-authentication-plugin=mysql\_native\_password --character-set-server=utf8 --collation-server=utf8\_general\_ci*

*environment:*

*- MYSQL\_ROOT\_PASSWORD=verysecret*

*- MYSQL\_DATABASE=iceland*

*- MYSQL\_USER=alexey*

*- MYSQL\_PASSWORD=Alexey*

Тогда, в конфигурации развернутого php yii2 проекта пропишем в файле config/db.php настройки конфигурации базы данных.

*<?php*

*return [*

*'class' => 'yii\db\Connection',*

*'dsn' => 'mysql:host=mysql;dbname=iceland',*

*'username' => 'alexey',*

*'password' => 'alexey',*

*'charset' => 'utf8',*

*];*

Создадим базу данных.

Выполним команду docker-compose run php yii migrate/create create\_city\_table, тем самым создав миграцию для таблицы create\_city\_table, чтобы впоследствии была возможность использовать данную структуру в mysql базе. Приведем пример создания миграции для одной из таблиц:

*<?php*

*use yii\db\Migration;*

*/\*\**

*\* Handles the creation of table `{{%chapter}}`.*

*\*/*

*class m201117\_113315\_create\_chapter\_table extends Migration*

*{*

*/\*\**

*\* {@inheritdoc}*

*\*/*

*public function safeUp()*

*{*

*$this->createTable('{{%chapter}}', [*

*'id' => $this->primaryKey()->notNull(),*

*'idCity' => $this->integer()->comment('Название города'),*

*'text' => $this->string(100)->notNull()->comment('Текст'),*

*'photoPath' => $this->string()->notNull()->comment('Путь к изображению'),*

*'link' => $this->string()->notNull()->comment('Ссылка в приложении'),*

*]);*

*$this->addForeignKey(*

*'fk-chapter-idCity',*

*'chapter',*

*'idCity',*

*'city',*

*'id',*

*'NO ACTION',*

*'NO ACTION'*

*);*

*}*

*/\*\**

*\* {@inheritdoc}*

*\*/*

*public function safeDown()*

*{*

*$this->dropForeignKey(*

*'fk-chapter-idCity',*

*'chapter'*

*);*

*$this->dropTable('{{%chapter}}');*

*}*

*}*

Накатить миграцию можно при помощи команды: docker-compose run php yii migrate/up

**Разработка структуры**

Всего база данных содержит 12 таблиц: Достопримечательность, Кухня, Путешествие, Жилье, Раздел главной страницы, Комментарий, Пользователь, Раздел, Фотография, Заметка, Город, Раздел города. Таблицы БД, а также связи между ними показаны на фотографии ниже.

Diagram

Description automatically generated

**Разработка моделей**

После того, как в удаленной базе данных появилась таблица (например, при помощи миграции), есть возможность получить класс-модель соответствующей таблицы, благодаря yii code generator <http://localhost:1199/gii>. В качестве метода взаимодействия с базой данных используется Active Record. Покажем, как осуществляется получение класса:

Graphical user interface, text, application

Description automatically generated

Заполняем необходимые поля

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Копируем полученную из Yii2 Model Generator информацию в app\modules\v1\models\Chapter.php; Меняем строку namespace с указанием нового пути: namespace app\modules\v1\models;

Graphical user interface, text, application

Description automatically generated

Так выглядит код сгенерированной модели таблицы разделов:

*<?php*

*namespace app\modules\v1\models;*

*use Yii;*

*/\*\**

*\* This is the model class for table "chapter".*

*\**

*\* @property int $id*

*\* @property int|null $idCity Название города*

*\* @property string $text Текст*

*\* @property string $photoPath Путь к изображению*

*\* @property string $link Ссылка в приложении*

*\**

*\* @property City $idCity0*

*\*/*

*class Chapter extends \yii\db\ActiveRecord*

*{*

*/\*\**

*\* {@inheritdoc}*

*\*/*

*public static function tableName()*

*{*

*return 'chapter';*

*}*

*/\*\**

*\* {@inheritdoc}*

*\*/*

*public function rules()*

*{*

*return [*

*[['idCity'], 'integer'],*

*[['text', 'photoPath', 'link'], 'required'],*

*[['text'], 'string'],*

*[['photoPath', 'link'], 'string', 'max' => 255],*

*[['idCity'], 'exist', 'skipOnError' => true, 'targetClass' => City::className(), 'targetAttribute' => ['idCity' => 'id']],*

*];*

*}*

*/\*\**

*\* {@inheritdoc}*

*\*/*

*public function attributeLabels()*

*{*

*return [*

*'id' => 'ID',*

*'idCity' => 'Название города',*

*'text' => 'Текст',*

*'photoPath' => 'Путь к изображению',*

*'link' => 'Ссылка в приложении',*

*];*

*}*

*/\*\**

*\* Gets query for [[IdCity0]].*

*\**

*\* @return \yii\db\ActiveQuery*

*\*/*

*public function getIdCity0()*

*{*

*return $this->hasOne(City::className(), ['id' => 'idCity']);*

*}*

*}*

Повторим действия с написанием и применением миграций для всех таблиц базы данных. Получим классы при помощи генератора моделей.

Text

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Для заполнения Базы данных используем простую функцию написанного REST-API, подобно следующей:

*class AttractionController extends ApiController{*

*/\*\**

*\* Пример запроса:*

*\* http://127.0.0.1:1199/api/v1/attraction/create*

*\*/*

*public function actionCreate(){*

*$model = new Attraction();*

*$model->load(Yii::$app->request->getBodyParams(), '');*

*$model->save();*

*return $model;*

*}*

*}*

Тогда заполнить БД можно при помощи GET-запроса из POSTMAN, передав в тело нужные параметры. Приведем пример:

Graphical user interface, text, application, email

Description automatically generated

Таким образом, заполним всю базу данных и проверим наличие данных в adminer, например, для таблицы заметок:

Graphical user interface, text, application, email

Description automatically generated

**Разработка REST API**

**Особенности разработки REST API при помощи YII2**

После подключения Active Query, внутри контроллеров, осуществляющих обработку API запросов пользователя можно обратиться к определенной таблице при помощи Table::method. method – обертка над SQL запросом. Например: findOne([]), findAll([]) или model.save –  для коммита изменений.

Отдельного упоминания служит работа с пользователем.

Когда пользователь входит в систему: его пароль проходит через хэш-функцию и данный хэш сравнивается с находящимся в БД. В случае удачной аутентификации, пользователь получает от сервера access token, который сохраняется в Loсal Storage браузера. Токен позволят пользователю получить индивидуальные полномочия по отношению к комментариям (модификацию и удаление). Во время выхода пользователя из системы, у него погашается токен, запись о пользователе стирается из Local Storage.

**Документация к API**

Опишем API-методы для следующих классов:

1) Контроллер главной страницы

2) Город

3) Достопримечательности

4) Варианты проживания

5) Кухня

6) Способы передвижения

7) Пользователь

Базовый URL для всех методов: [*http://127.0.0.1:1199/api/v1/*](http://127.0.0.1:1199/api/v1/)

**Контроллер главной страницы**

*URL:* [*http://127.0.0.1:1199/api/v1/*maincontent](http://127.0.0.1:1199/api/v1/maincontent/)

Методы:

**1) Создание нового раздела главной страницы**

URL: /create

Тип запроса: POST

Параметры: в Body => name: String!, link: String!

*(ПРИМ. «!» соответствует обязательному параметру)*

**2) Получение названий разделов главной страницы**

URL: /names

Тип запроса: GET

Параметры: Нет

**3) Получение разделов главной страницы**

URL: /chapters

Тип запроса: GET

Параметры: Нет

**4) Получение заметок главной страницы**

URL: /notes

Тип запроса: GET

Параметры: Нет

**5) Получение текста информации главной страницы**

URL: /description

Тип запроса: GET

Параметры: Нет

**6) Получение фотографий главной страницы**

URL: /photo

Тип запроса: GET

Параметры: Нет

**7) Получение видео главной страницы**

URL: /getvideo

Тип запроса: GET

Параметры: Нет

**8) Изменение ссылки видео главной страницы**

URL: /addvideo

Тип запроса: GET

Параметры: link – строка, содержащая новую ссылку на видеоресурс

**Город**

*URL:* [*http://127.0.0.1:1199/api/v1/*city/](http://127.0.0.1:1199/api/v1/city/)

Методы:

**1) Создание нового города**

URL: /create

Тип запроса: POST

Параметры: в Body => name: String!, link: String!

**2) Получение города по латинскому названию**

URL: /getbyname

Тип запроса: GET

Параметры: engname (латинское название города): String!

**3) Получение названий городов**

URL: /names

Тип запроса: GET

Параметры: Нет

**4) Получение названия города по id**

URL: /name

Тип запроса: GET

Параметры: id (идентификатор города): Int!

**5) Получение списка разделов города по id города**

URL: /chapters

Тип запроса: GET

Параметры: idCity (идентификатор города): Int!

**6) Получение списка заметок по id города**

URL: /getnotes

Тип запроса: GET

Параметры: idCity (идентификатор города): Int!

**7) Получение фотографий города по id города**

URL: /photo

Тип запроса: GET

Параметры: idCity (идентификатор города): Int!

**8) Получение главной фотографии города по id города**

URL: /mainphoto

Тип запроса: GET

Параметры: idCity (идентификатор города): Int!

**Достопримечательности**

*URL:* [*http://127.0.0.1:1199/api/v1/*attraction/](http://127.0.0.1:1199/api/v1/attraction/)

Методы:

**1) Создание новой достопримечательности**

URL: /create

Тип запроса: POST

Параметры: в Body => name: String!

**2) Получение названий достопримечательностей**

URL: /names

Тип запроса: GET

Параметры: Нет

**3) Получение списка разделов достопримечательностей**

URL: /chapters

Тип запроса: GET

Параметры: Нет

**4) Получение заметки по названию достопримечательности**

URL: /note

Тип запроса: GET

Параметры: attractionname (Название достопримечательности): String!

**5) Получение фотографий достопримечательностей**

URL: /photo

Тип запроса: GET

Параметры: Нет

**6) Получение комментарий к странице достопримечательностей**

URL: /coments

Тип запроса: GET

Параметры: Нет

**7) Создание нового комментария**

URL: /new\_comment

Тип запроса: POST

Параметры: accessToken: String!, text: String!

**8) Редактирование существующего комментария**

URL: /edit\_comment

Тип запроса: POST

Параметры: accessToken: String!,

id (идентификатор комментария): String!,

text: String!

**9) Удаление комментария**

URL: /delete\_comment

Тип запроса: POST

Параметры: accessToken: String!,

id (идентификатор комментария): String!,

Для следующих 3 классов укажем лишь url, т.к суть методов для них схожа. А затем опишем методы.

**Варианты проживания**

*URL:* [*http://127.0.0.1:1199/api/v1/*housing/](http://127.0.0.1:1199/api/v1/housing/)

**Варианты передвижения**

*URL:* [*http://127.0.0.1:1199/api/v1/*trip/](http://127.0.0.1:1199/api/v1/trip/)

**Кухня**

*URL:* [*http://127.0.0.1:1199/api/v1/*kitchen/](http://127.0.0.1:1199/api/v1/kitchen/)

Методы для перечисленных 3 классов:

**1) Создание нового (**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /create

Тип запроса: POST

Параметры: в Body => name: String!

**2) Получение названий**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /names

Тип запроса: GET

Параметры: Нет

**3) Получение списка разделов**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /chapters

Тип запроса: GET

Параметры: Нет

**4) Получение заметки по названию**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /notes

Тип запроса: GET

Параметры: housing (Название варианта проживания): String! |

trip (Название варианта передвижения): String! |

kitchen (Название раздела кухни): String!

**5) Получение фотографий**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /photo

Тип запроса: GET

Параметры: Нет

**6) Получение главной фотографии страницы**

**(Варианта проживания |**

**Передвижения|**

**Кухни)**

URL: /mainphoto

Тип запроса: GET

Параметры: Нет

**Пользователь**

*URL:* [*http://127.0.0.1:1199/api/v1/*user/](http://127.0.0.1:1199/api/v1/user/)

Методы:

**1) Получение списка всех пользователей**

URL: /all

Тип запроса: GET

Параметры: Нет

**2) Получение access token для входа в систему пользователя**

URL: /signin, /signup

Тип запроса: POST

Параметры: login: String!,

password: String!

**3) Выход пользователя из системы**

URL: /exit

Тип запроса: POST

Параметры: accessToken: String!

**Клиентские HTTP запросы**

**Axios JS - средство для отправки AJAX запросов на сервер**

Для выполнения http-запросов из Vue.js была выбрана библиотека Axios. При помощи axios.get(url).then(promise), можно совершить AJAX get-запрос по заданному url. AJAX – означает без перезагрузки страницы.

При помощи axios.post(url, body-dict).then(promise), можно совершить AJAX post-запрос по заданному url. body-dict представляет собой словарь, содержащий пары ключ значения переданных параметров.

**Взаимодействие клиентской и серверной части приложения**

Приведем пример для метода регистрации пользователя в системе со стороны клиента (Vue JS). Создается переменная instance, которая содержит в себе адрес сервера. Далее, к этому серверу совершается POST запрос с передачей параметров в теле и обрабатывается ответ через промис.

*signUp(e){*

*let thisEnv = this*

*var token = null*

*const instance = Axios.create({*

*baseURL: 'http://127.0.0.1:1199/api/v1'*

*})*

*instance.post('/user/signup', {"login": thisEnv.email, "password": thisEnv.password}).then(function(responseToken){*

*token = responseToken.data.token*

*if (responseToken.data.statusText == undefined){*

*User.login({*

*email: thisEnv.email,*

*accessToken: token*

*})*

*e.preventDefault()*

*thisEnv.$forceUpdate()*

*thisEnv.parentRefresh()*

*} else {*

*thisEnv.errors.email = 'Пользователь с таким логином уже существует!'*

*thisEnv.errors.password = ''*

*thisEnv.password = ''*

*document.getElementById('gridCheck1').checked = false*

*e.preventDefault()*

*}*

*})*

*},*

**Скриншоты**

Graphical user interface, website

Description automatically generated

Graphical user interface, website

Description automatically generated

**Graphical user interface, text, website

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Заключение**

В данной работе мы поэтапно разработали приложение «Путеводитель по Исландии». Описали будущие страницы, а затем сверстали их при помощи HTML, CSS, Bootstrap. Использовали Vue JS для написания клиентской части приложения. Развернули серверную часть приложения при помощи PHP Yii2. Написали структуру таблиц MySQL, произвели миграции и оформили в виде REST API на стороне сервера запросы к этой базе. Заполнили базу запросами написанного API. Соединили клиентскую и серверную часть путем совершения HTTP запросов по написанному API, используя библиотеку Axios.

**Список литературы**

1) <https://uguide.ru/tablica-osnovnykh-tegov-html-s-primerami>

2)<https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics>

3) <https://proglib.io/p/vue-twitter>

4) <https://tproger.ru/video/vue-js-for-beginners-videos/>

5) https://starkovden.github.io/what-is-rest-api.html

6) <https://www.yiiframework.com/doc/guide/2.0/en/start-installation#installing-composer>

7) https://ru.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html