МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»

(БГТУ им. В.Г.Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Технологии Web-программирования

РГЗ

**Разработка веб-сервиса «Касса ЖД билетов»**

Выполнил: студент группы ВТ-41

Клесов М.И.

Проверил:

Картамышев С.В.

Белгород 2020

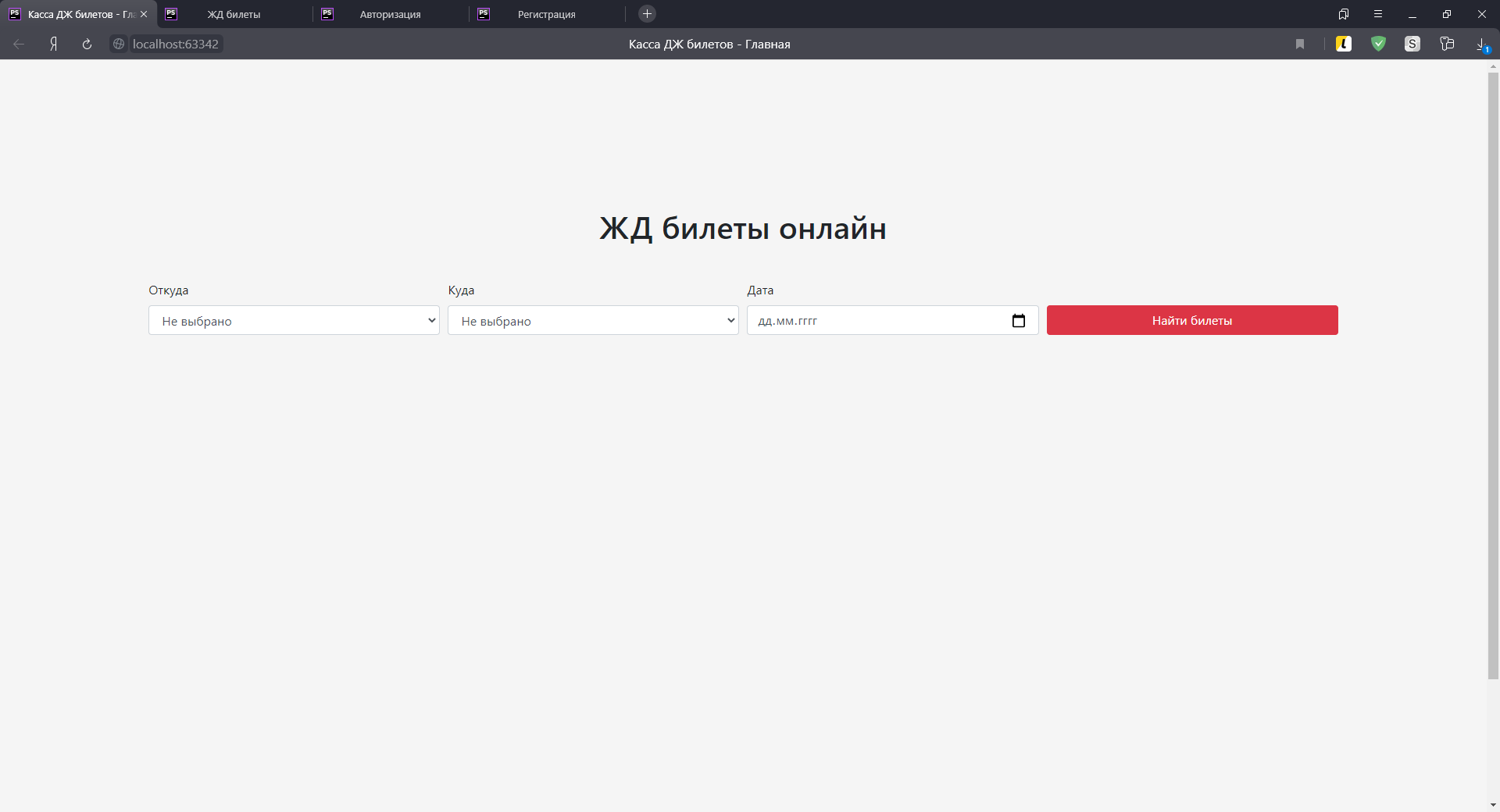
**Цель:** разработать веб-приложение по выбранной теме, познакомится с вёрсткой страниц с помощью HTML и CSS, основами разработки на языке JavaScript, программированием backend-а на языке PHP, разработать БД, овладеть механизмами взаимодействия сервера и базы данных и взаимодействия REST API сервера и frontend-а.

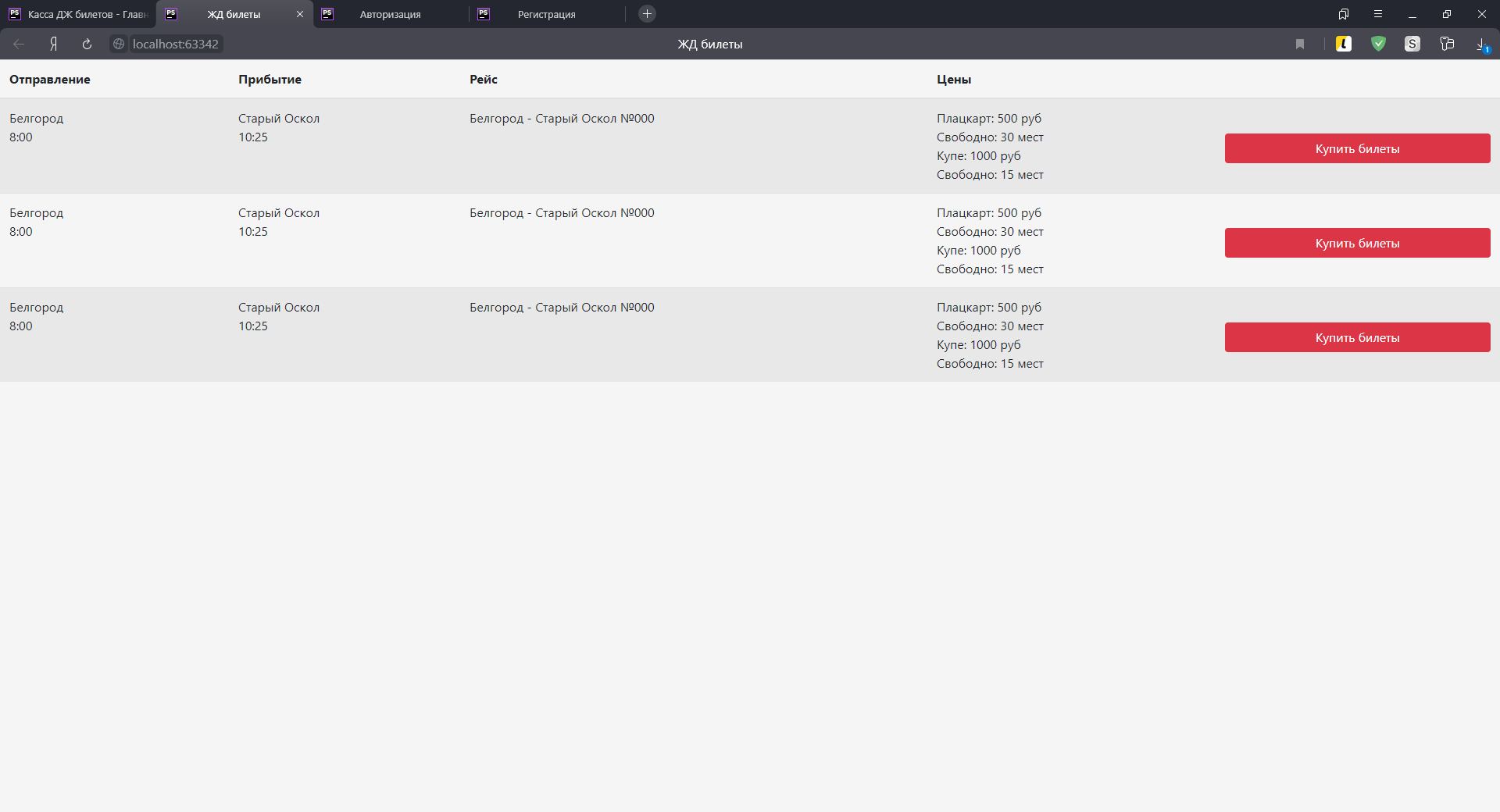
**Задания к работе**

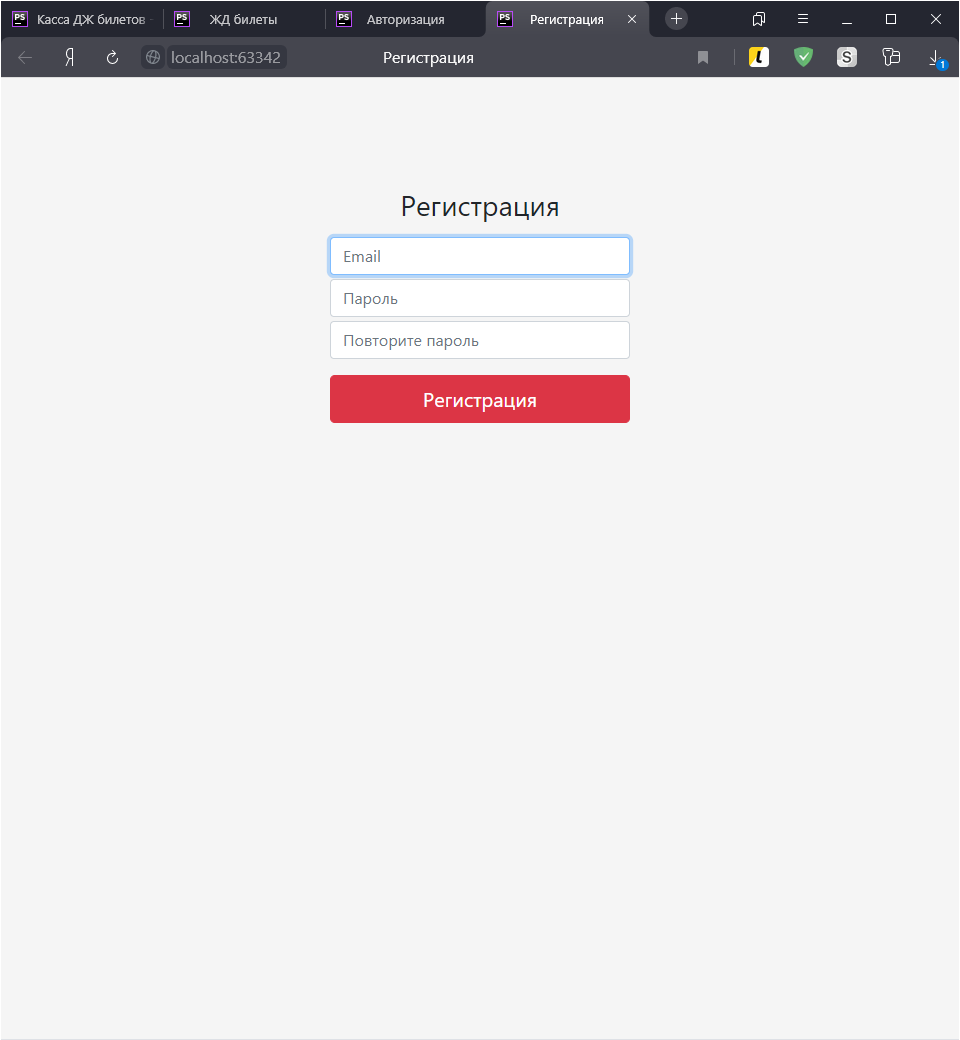
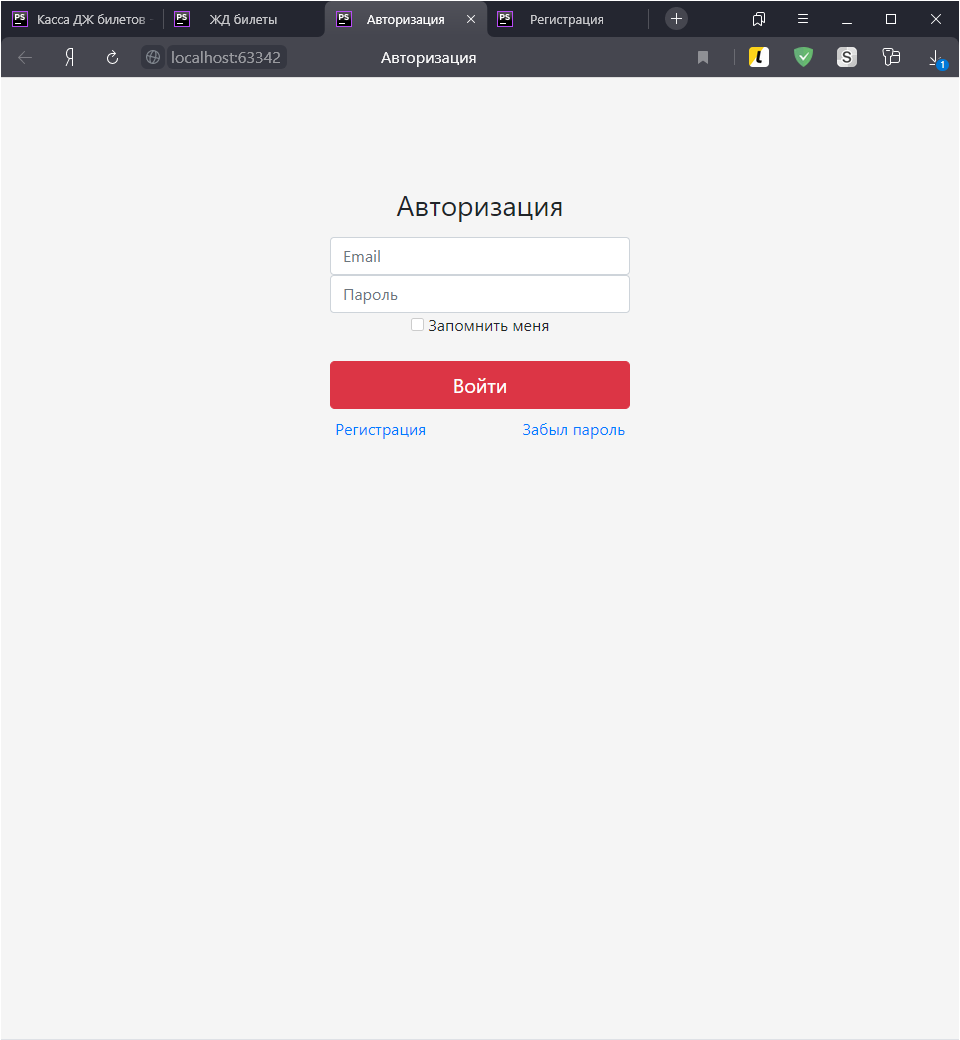
1. Разработать макет и верстку шаблонов web-приложения с помощью языков HTML и CSS;
2. Перенести макет на фреймворк Vue JS;
3. Развернуть базовое приложение Yii2 App Basic и добавить несколько тестовых контроллеров со статичными данными;
4. Разработать и спроектировать БД, добавить соответствующие модели в backend-приложение;
5. Разработка REST API;
6. Реализовать взаимодействие frontend-а и REST API с помощью Ajax запросов.

**Выполнение**

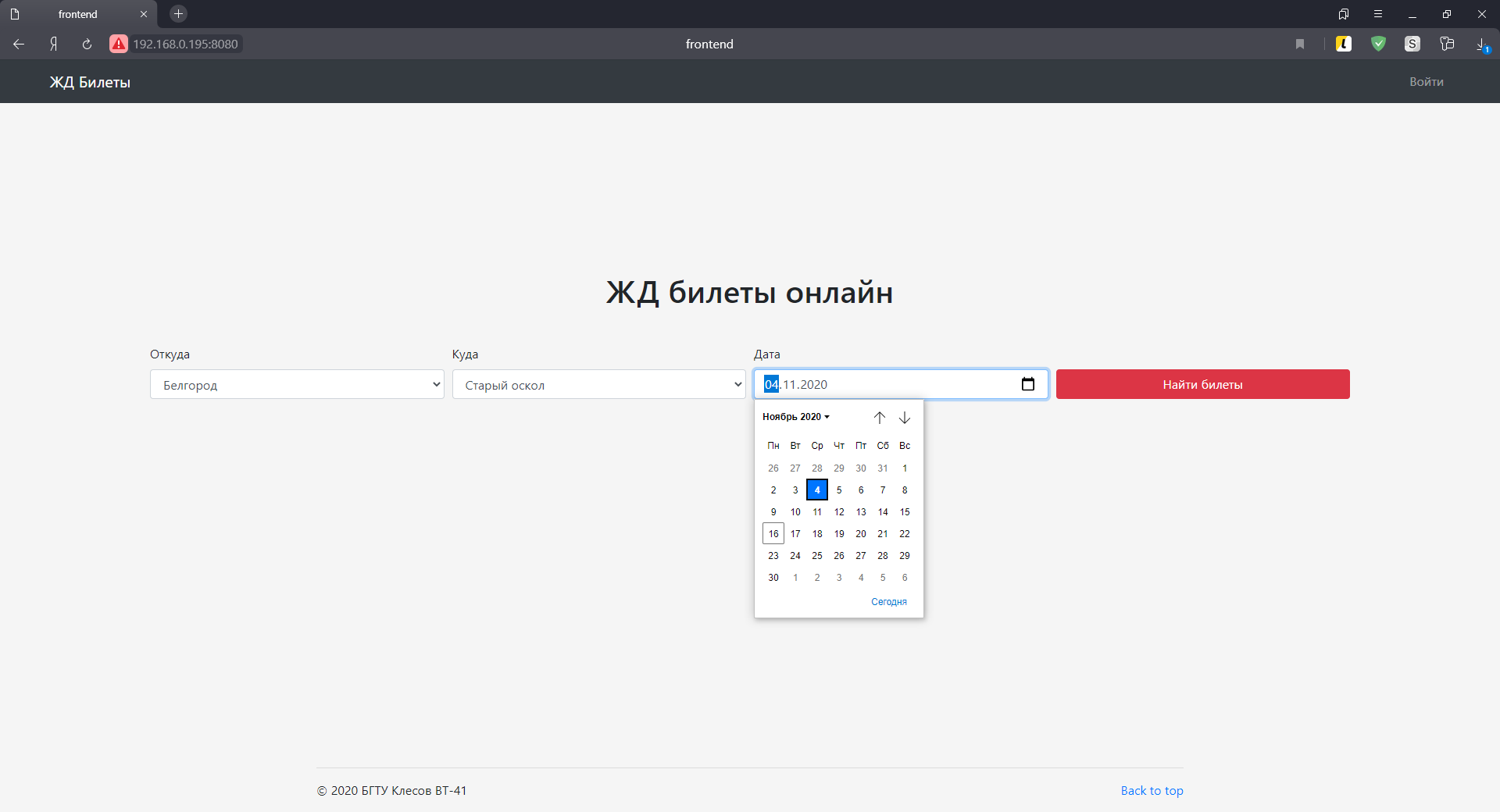
1. Разработаны начальные макеты страниц: главная, контентная, авторизации и регистрации с использованием Bootstrap - фреймворка для HTML и CSS:

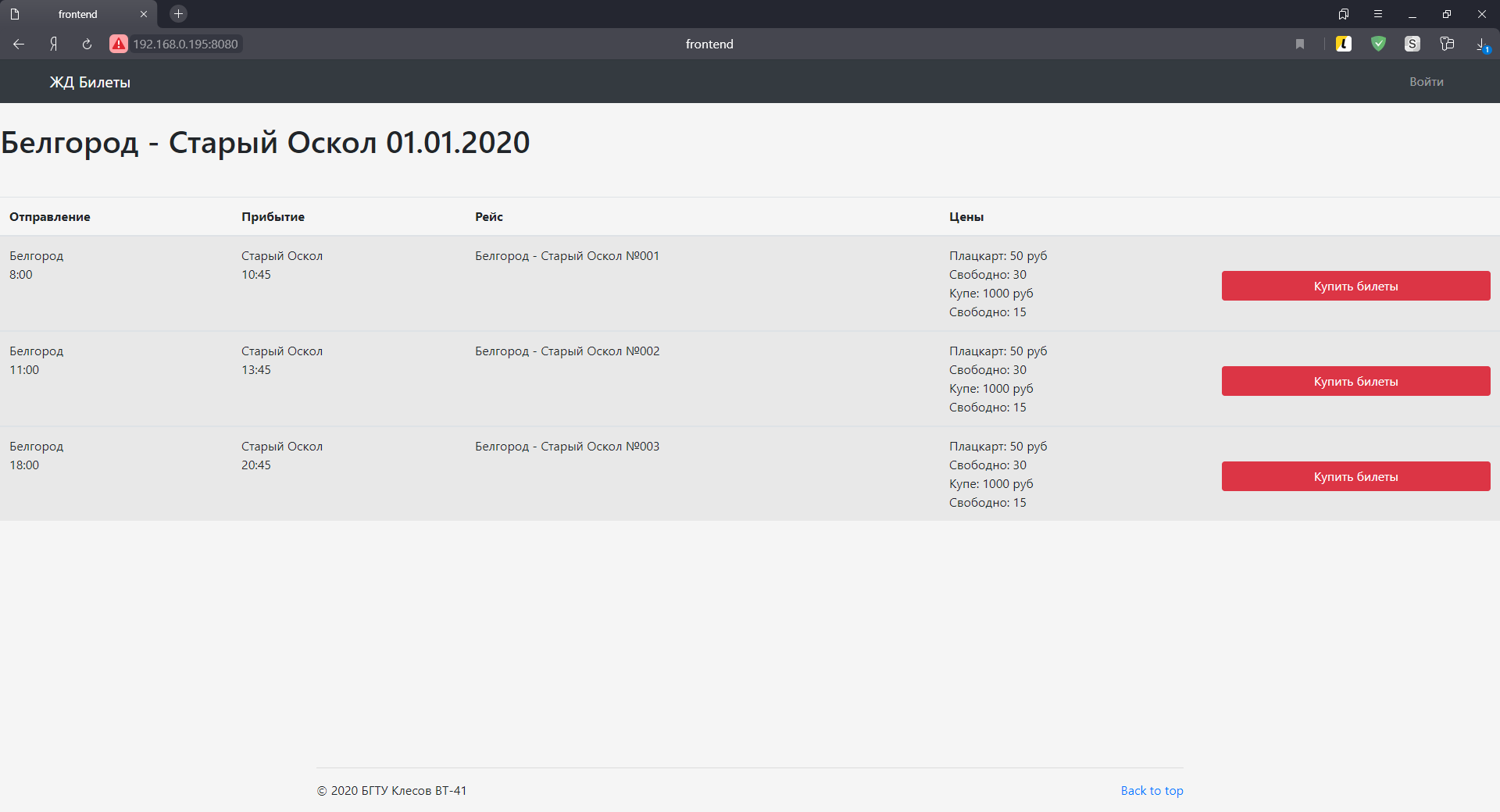


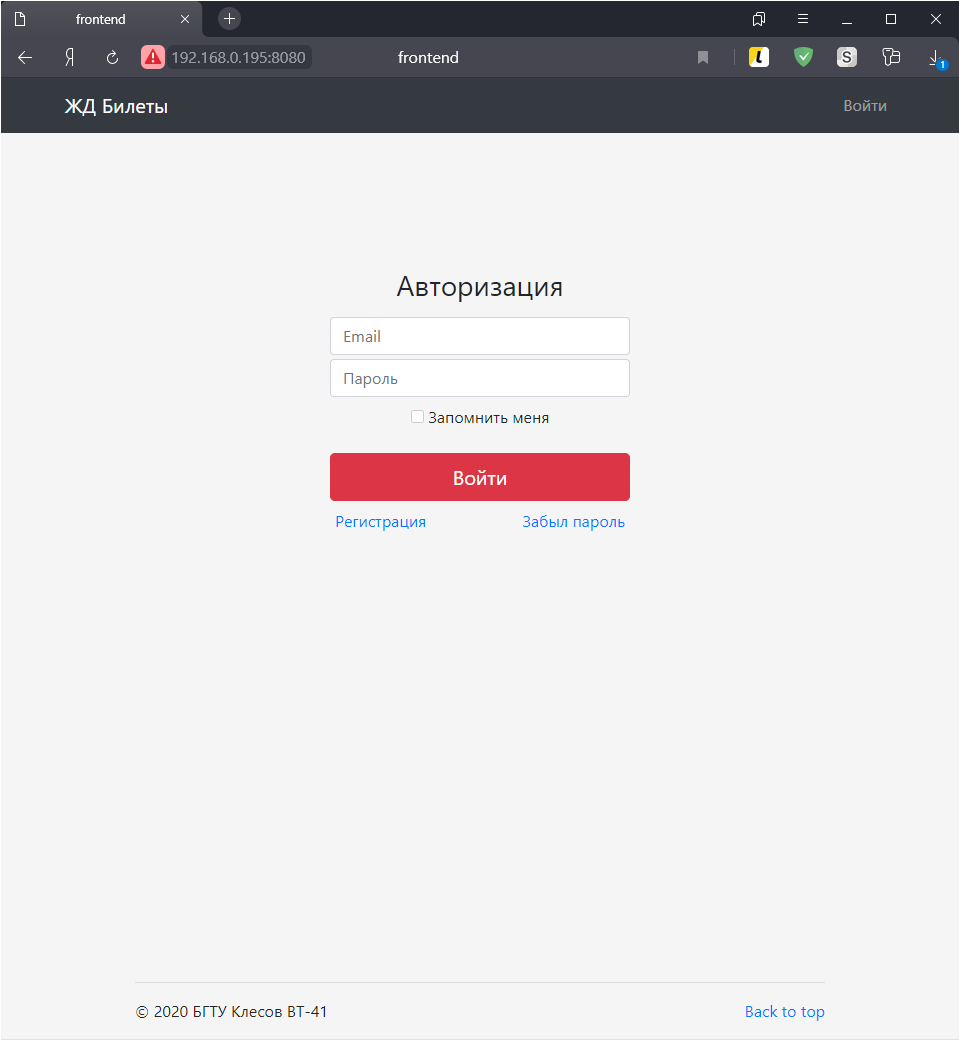
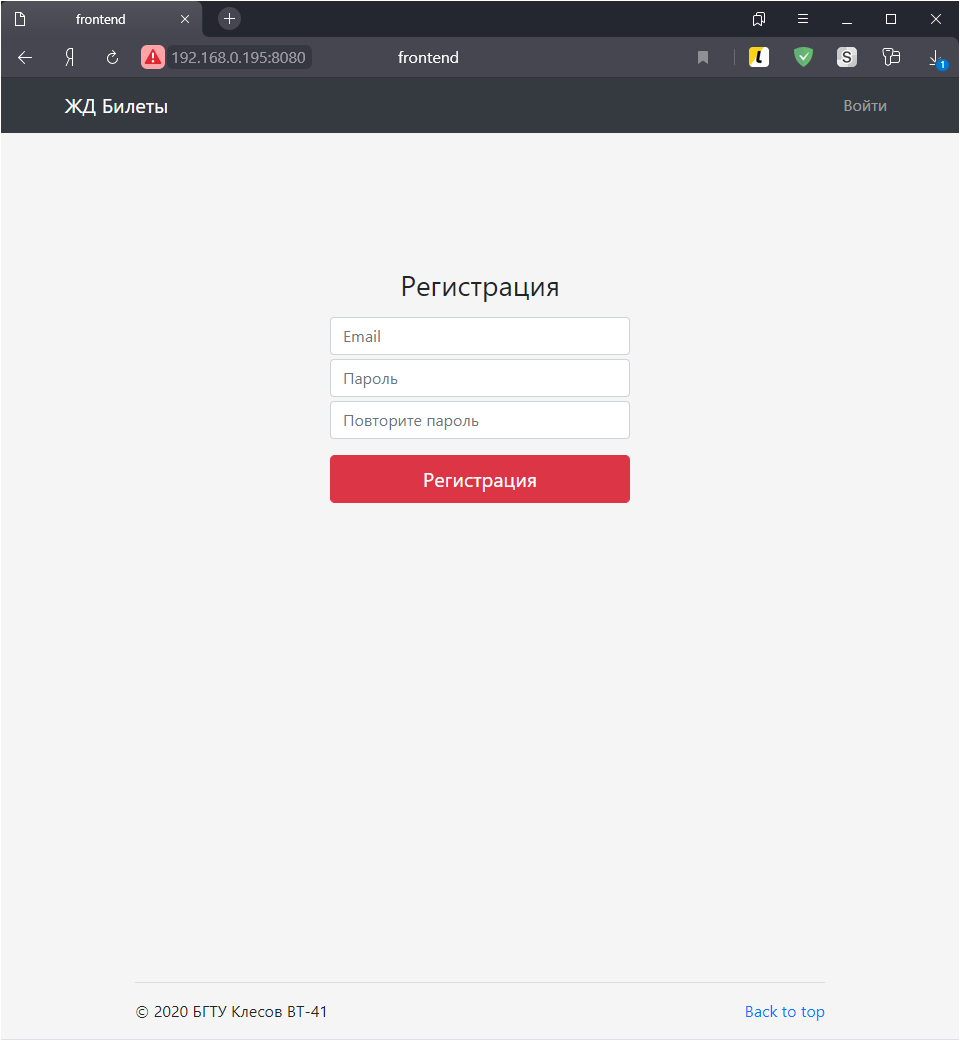




1. Макеты страниц из предыдущего пункта были доработаны и перенесены в компоненты фреймворка Vue JS. Также реализованы переходы между страницами, добавлены футер и навбар:

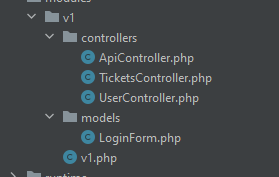




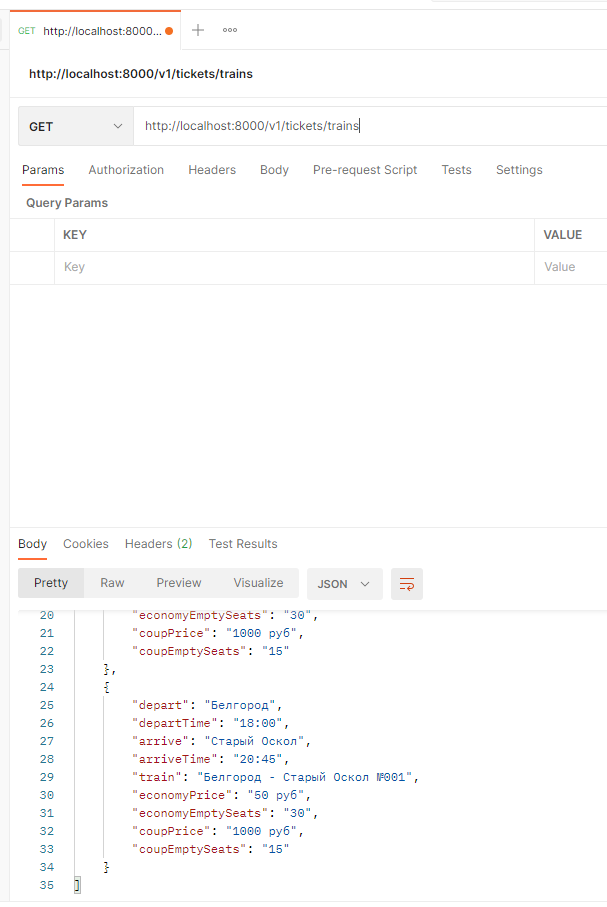
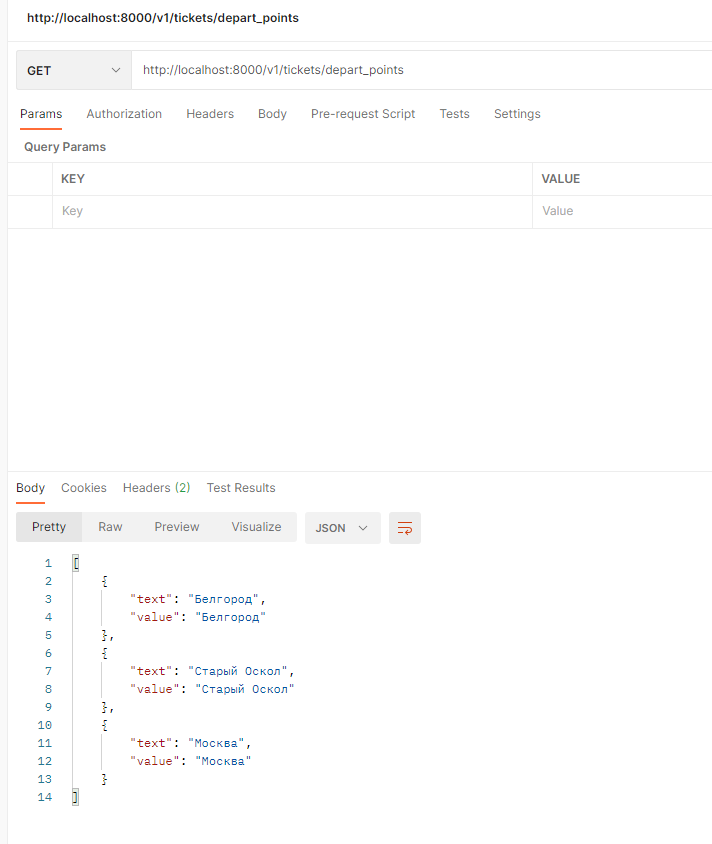
 

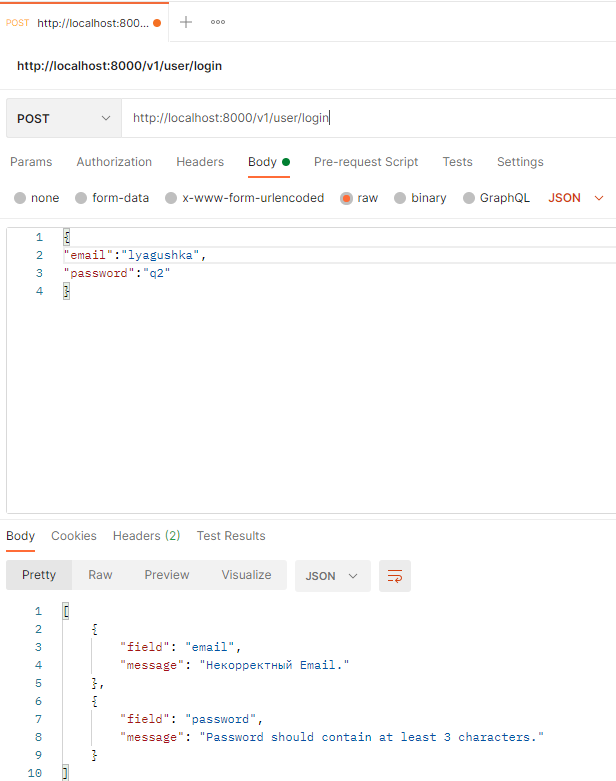
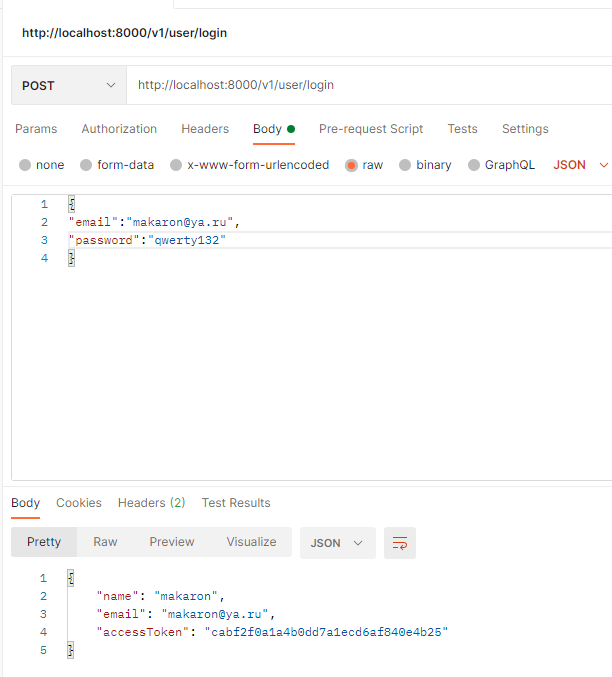
1. Развёрнули базовое приложение Yii2 App Basic с помощью команды composer create-project --prefer-dist yiisoft/yii2-app-basic basic. Затем это приложение было настроено для работы (отредактировали файлы конфигураций и добавили поддержку CORS запросов), созданы тестовые контроллеры, возвращающие константные данные. Осуществили передачу константных данных в frontend посредством REST API.

Созданные контроллеры:

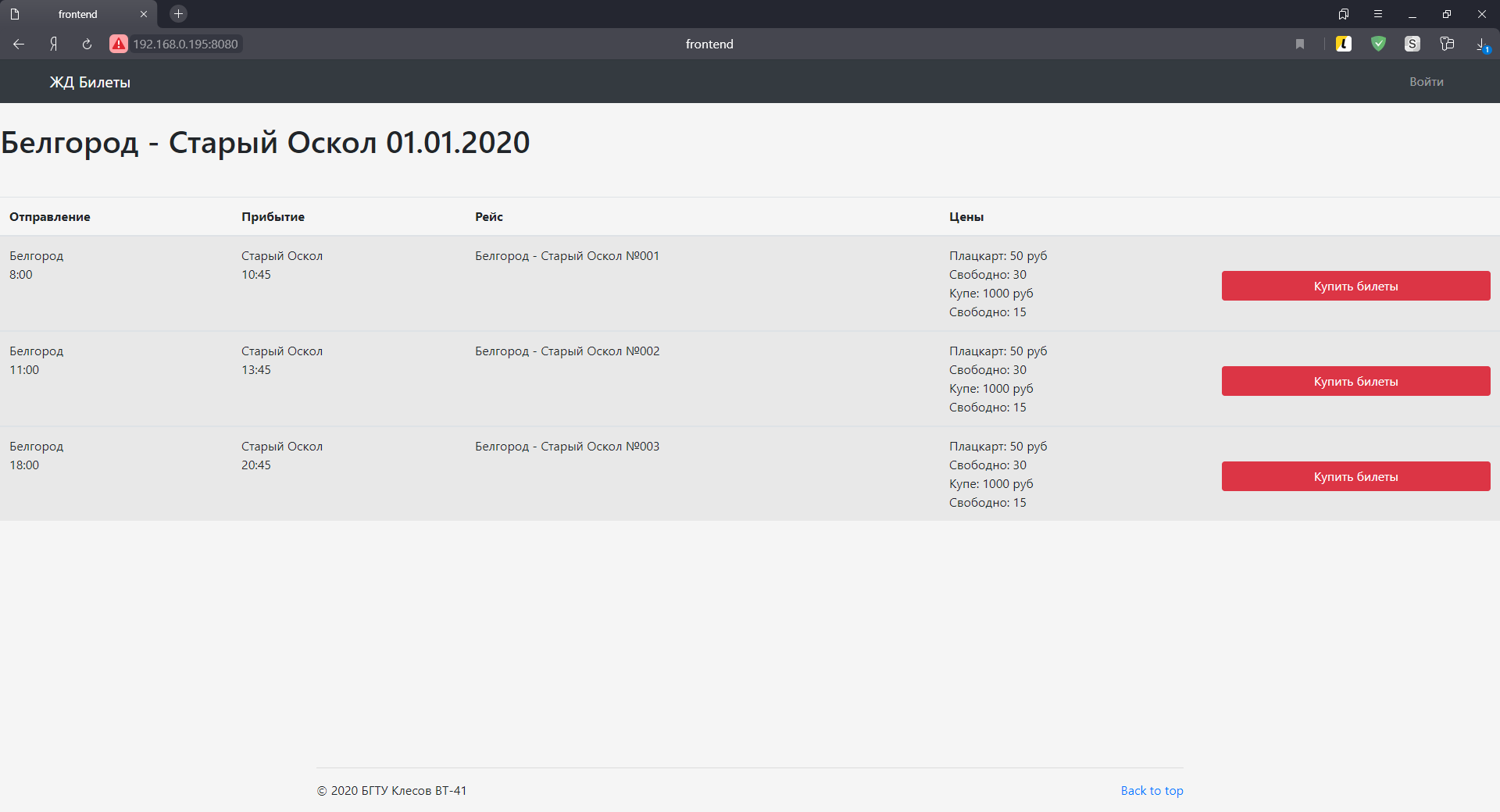
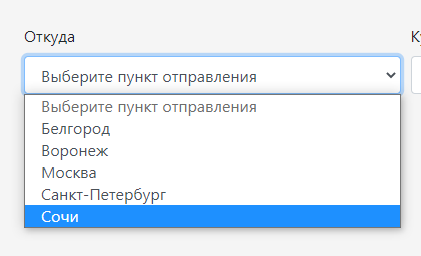


Тестирование API:

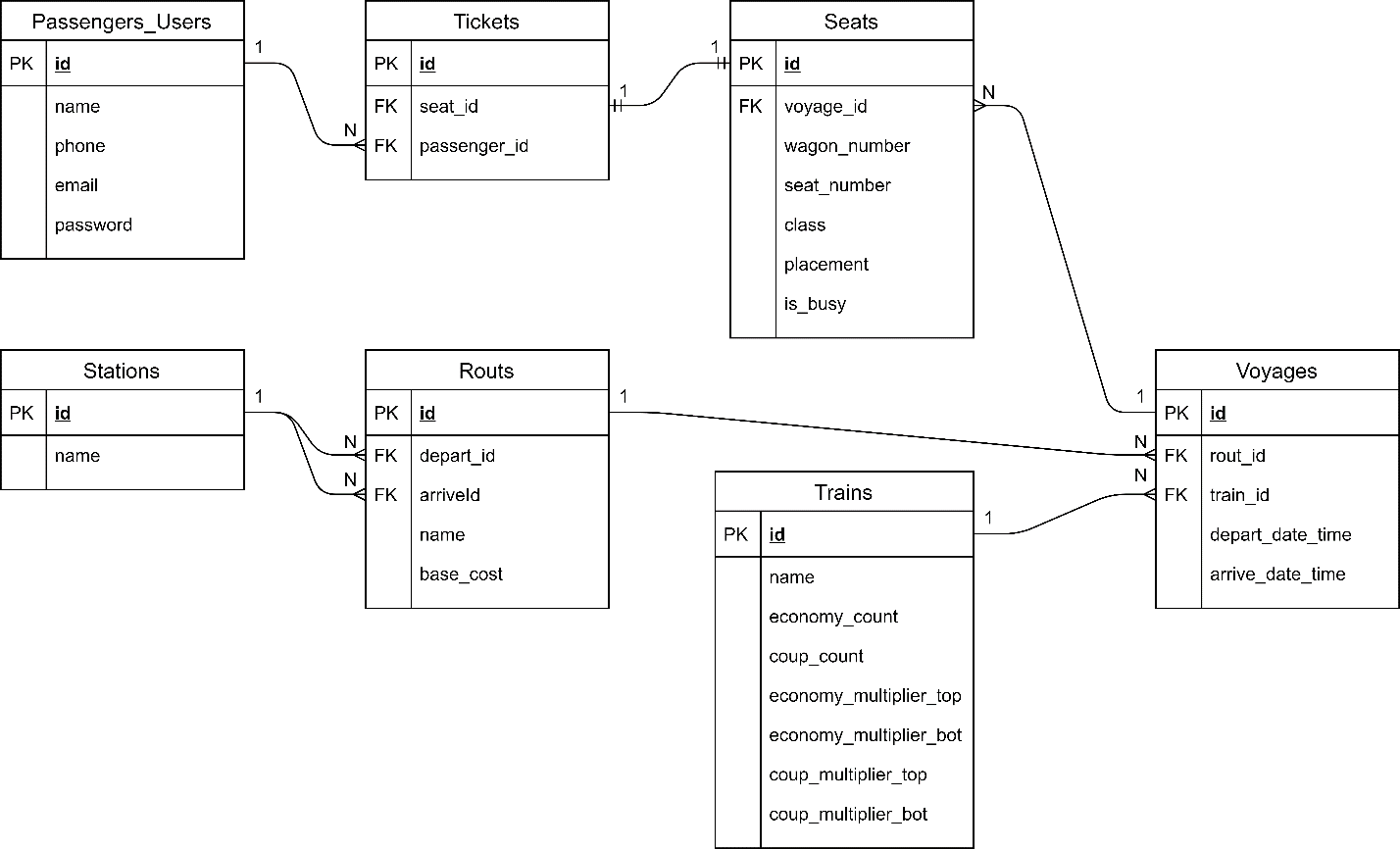




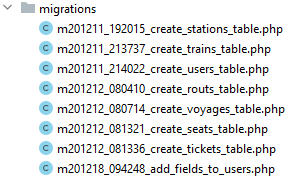
Результат в frontend-приложении:



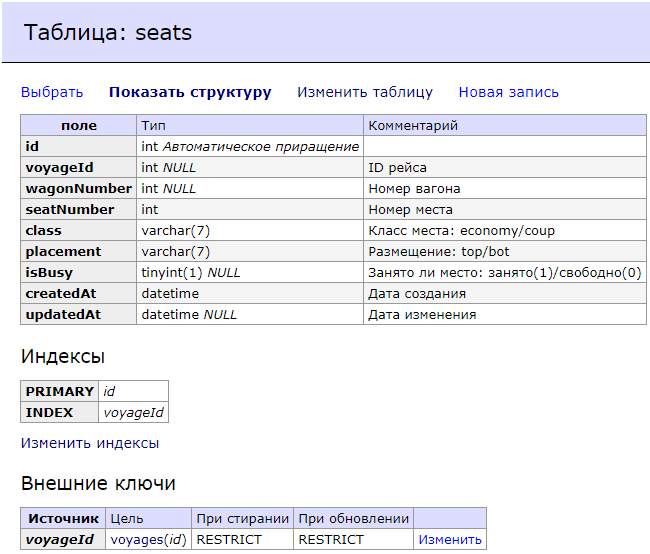
1. Спроектировали схему базы данных:

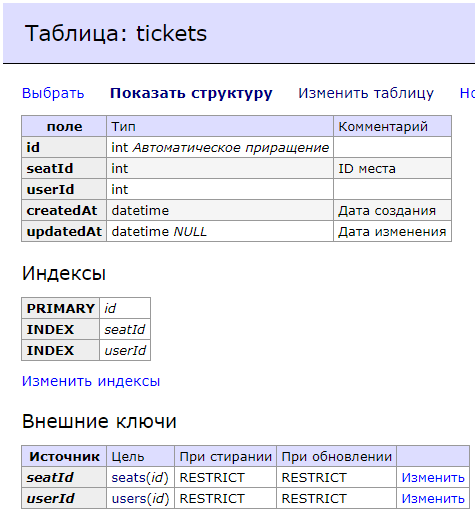
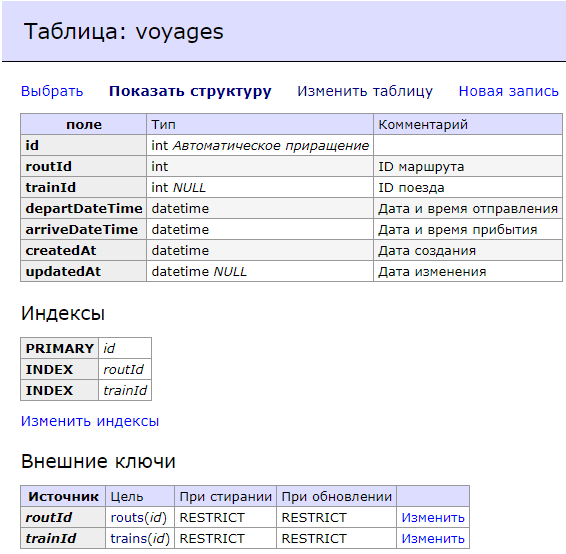


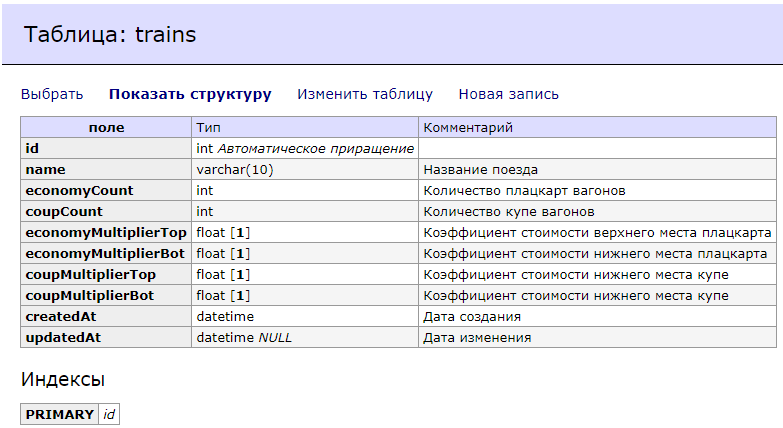
В качестве СУБД была выбрана MySQL, настроена конфигурация докера для работы с БД и созданы миграции для генерирования БД:



Полученная структура БД:

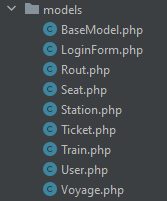






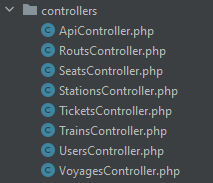


Созданные модели, соответствующие сущностям БД:



1. Разработали REST API, а также дополнили функционал некоторых моделей БД.

Разработанные контроллеры:



Документация к PAI:

**Общее описание к методам:**

Базовый URL: <http://192.168.0.195::1149/v1>

**Контроллеры:**

Контроллер маршрутов RoutsController

URL: /routs

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionRouts | Возвращает все маршруты | Нет |  |  |
| /routs |
| actionRout | Возвращает маршрут по его id | id | integer | Да |
| /rout |

Контроллер посадочных мест SeatsController

URL: /seats

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionVoyage\_empty\_seats | Возвращает свободные посадочные места для рейса voyageId | voyageId | Integer | Да |
| /voyage\_empty\_seats |

Контроллер станций StationsController

URL: /stations

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionStation | Возвращает станцию по ей id | id | Integer | Да |
| /station |
| actionAll\_departs | Возвращает все станции отправления | Нет |  |  |
| /all\_departs |
| actionAll\_arrives | Возвращает все станции прибытия | Нет |  |  |
| /all\_arrives |

Контроллер билетов TicketsController

URL: /tickets

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionUser\_tickets | Возвращает билеты пользователя по его id | userId | Integer | Да |
| /user\_tickets |
| actionVoyage\_tickets | Возвращает билеты рейса по его id | vouageId | Integer | Да |
| /voyage\_tickets |
| actionTickets | Возвращает все билеты | Нет |  |  |
| /tickets |

Тип запроса: POST

Действия:

|  |  |  |
| --- | --- | --- |
| Название/url | Описание | Данные |
| actionBuy | Создаёт билет с привязкой к пользователю и посадочному месту по их id, устанавливает посадочное место как занятое | {userId, seatId} |
| /buy |

Контроллер поездов TrainsController

URL: /trains

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionVoyage\_train | Возвращает поезд для рейса по id рейса | voyageId | Integer | Да |
| /voyage\_train |
| actionTrains | Возвращает все поезда | Нет |  |  |
| /trains |

Контроллер пользователей UsersController

URL: /users

Тип запроса: GET

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionLogin | Выполняет авторизацию пользователя по его email и password\_md5. Возвращает соответствующего пользователя или исключение с кодом 406 – неверный логин или пароль | email  password\_md5 | Integer  String(32) | Да  Да |
| /login |

Тип запроса: POST

Действия:

|  |  |  |
| --- | --- | --- |
| Название/url | Описание | Данные |
| actionRegistration | Создаёт пользователя с указанным email и password\_md5, если пользователя с таким email не существует, иначе – исключение с кодом 406 | {email, password\_md5} |
| /registration |

Тип запроса: PUT

Действия:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionChange | Редактирует указанные атрибуты пользователя по его id | id  name  phone  email  password\_md5 | Integer  String(128)  String(32)  String(128)  String(32) | Да  Нет  Нет  Нет  Нет |
| /change |

Контроллер рейсов VoyagesController

URL: /voyages

Тип запроса: GET

Действия:

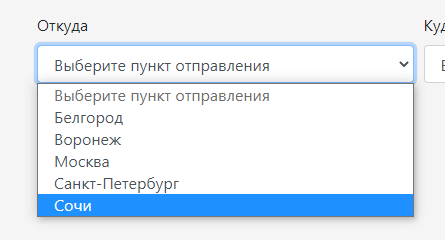
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название/url | Описание | Параметры | | |
| параметр | тип | обязательный |
| actionVoyage | Возвращает рейс по его id | id | Integer | Да |
| /voyage |

1. Для работы с AJAX запросами была выбрана библиотека Axios, представляющая собой HTTP-клиент, основанный на промисах и предназначенный для браузеров и Node.js.

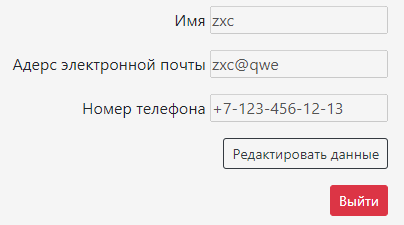
В ходе работы было реализовано взаимодействие фронтенда с REST API с помощью кроссдоменных HTTP запросов.

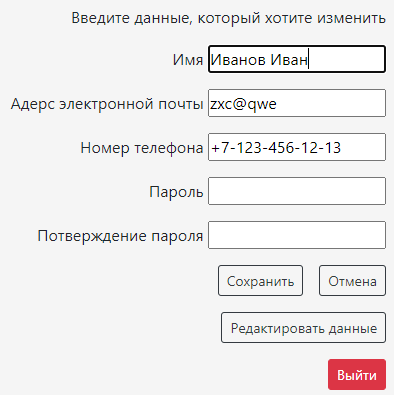
Примеры взаимодействия:

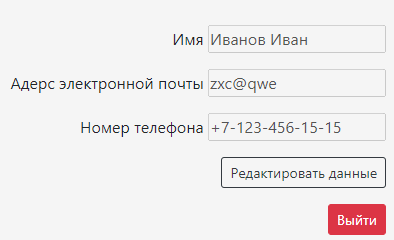
* Получение станций:



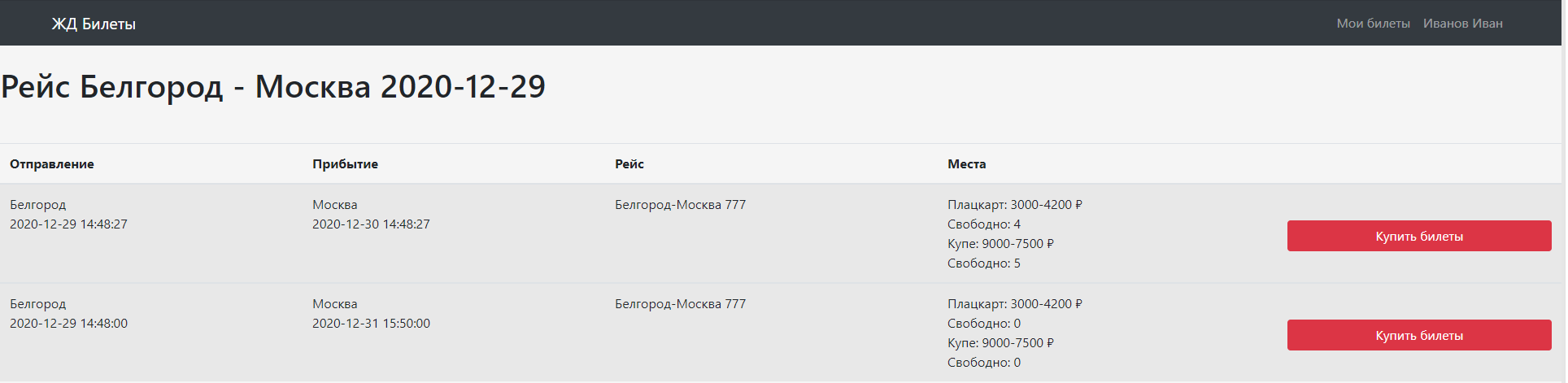
* Редактирование данных пользователя:







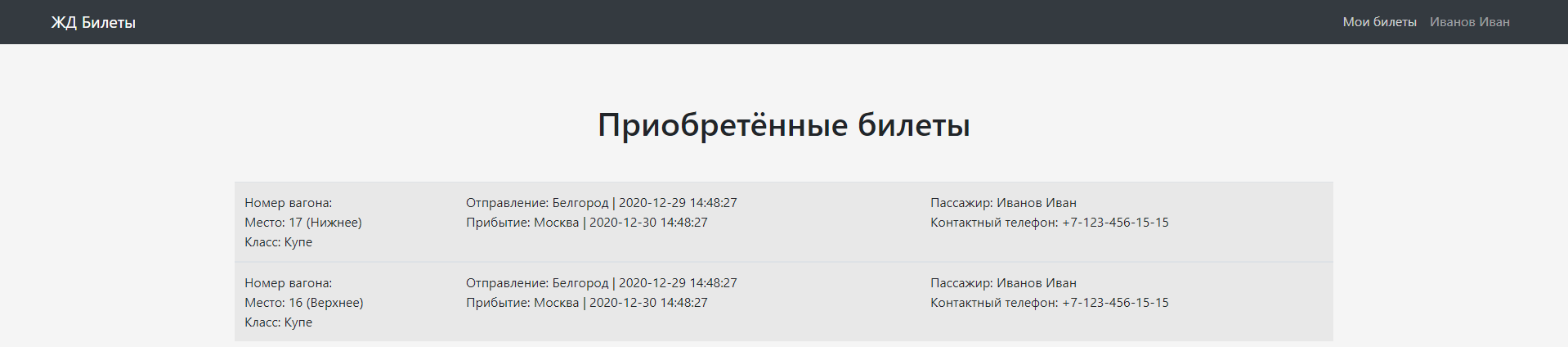
* Получение рейсов по заданным параметрам (станции и дата отправления):



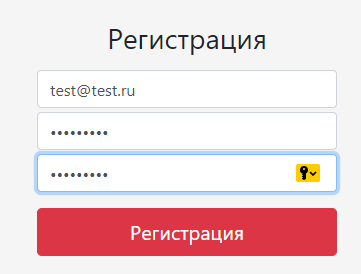
* Получение билетов для выбранного маршрута:

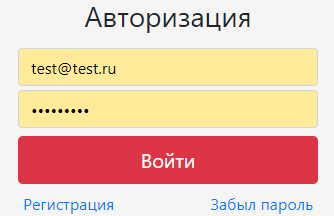


* Покупка билетов (свободное место пропадает из списка выше и в «Мои билеты» появляется приобретённый билет):



* Регистрация и авторизация:

**Приложение.**

Пример кода компонента Vue:

<template>  
 <div>  
 <h1>Рейс {{depart.name}} - {{arrive.name}} {{query.date}}</h1>  
 <table class="table table-striped my-5">  
 <thead>  
 <tr>  
 <th scope="col">Отправление</th>  
 <th scope="col">Прибытие</th>  
 <th scope="col">Рейс</th>  
 <th scope="col">Места</th>  
 <th scope="col"></th>  
 </tr>  
 </thead>  
 <tbody v-for="item in items">  
 <RoutingTableRow v-bind:voyage="item"></RoutingTableRow>  
 </tbody>  
 </table>  
 </div>  
</template>  
  
<script>  
import RoutingTableRow from "@/components/RoutingTableRow";  
export default {  
 name: "RoutingTable",  
 components: {RoutingTableRow},  
 data() {  
 return {  
 query: this.$route.query,  
 items: [],  
 depart: null,  
 arrive: null  
 }  
 },  
 created: function () {  
 this.$http.get('/stations/station', {params: {id: this.query.departId}})  
 .then((response) => { this.depart = response.data })  
 this.$http.get('/stations/station', {params: {id: this.query.arriveId}})  
 .then((response) => { this.arrive = response.data })  
 this.$http.get('/voyages/voyages',  
 {params: { departId: this.query.departId,  
 arriveId: this.query.arriveId,  
 depart\_date: this.query.date}})  
 .then((response) => { this.items = response.data })  
 }  
}  
</script>

Пример кода миграции:

<?php  
  
use yii\db\Migration;  
  
*/\*\*  
 \* Handles the creation of table `{{%voyages}}`.  
 \*/*class m201212\_080714\_create\_voyages\_table extends Migration  
{  
 */\*\*  
 \* Рейсы  
 \* {@inheritdoc}  
 \*/* public function safeUp()  
 {  
 $this->createTable('{{%voyages}}', [  
 'id' => $this->primaryKey(),  
 'routId' => $this->integer()->notNull()->comment('ID маршрута'),  
 'trainId' => $this->integer()->comment('ID поезда'),  
 'departDateTime' => $this->dateTime()->notNull()->comment('Дата и время отправления'),  
 'arriveDateTime' => $this->dateTime()->notNull()->comment('Дата и время прибытия'),  
 'createdAt' => $this->dateTime()->comment('Дата создания'),  
 'updatedAt' => $this->dateTime()->comment('Дата изменения')  
 ]);  
 $this->addForeignKey('fk\_routId', '{{%voyages}}', 'routId', '{{%routs}}', 'id');  
 $this->addForeignKey('fk\_trainId', '{{%voyages}}', 'trainId', '{{%trains}}', 'id');  
 }  
  
 */\*\*  
 \* {@inheritdoc}  
 \*/* public function safeDown()  
 {  
 $this->dropForeignKey('fk\_routId', '{{%voyages}}');  
 $this->dropForeignKey('fk\_trainId', '{{%voyages}}');  
 $this->dropTable('{{%voyages}}');  
 }  
}

Пример кода модели:

<?php  
  
namespace app\modules\v1\models;  
  
use app\modules\v1\models\BaseModel;  
use Yii;  
  
*/\*\*  
 \* This is the model class for table "tickets".  
 \*  
 \* @property int $id  
 \* @property int $seatId ID места  
 \* @property int $userId  
 \* @property string $createdAt Дата создания  
 \* @property string|null $updatedAt Дата изменения  
 \*  
 \* @property Seat $seat  
 \* @property User $user  
 \* @property Voyage $voyage  
 \* @property Train $train  
 \*/*class Ticket extends BaseModel  
{  
 */\*\*  
 \* {@inheritdoc}  
 \*/* public static function tableName()  
 {  
 return 'tickets';  
 }  
  
 */\*\*  
 \* {@inheritdoc}  
 \*/* public function rules()  
 {  
 return [  
 [['seatId', 'userId'], 'required'],  
 [['seatId', 'userId'], 'integer'],  
 [['createdAt', 'updatedAt'], 'safe'],  
 [['seatId'], 'exist', 'skipOnError' => true, 'targetClass' => Seat::class, 'targetAttribute' => ['seatId' => 'id']],  
 [['userId'], 'exist', 'skipOnError' => true, 'targetClass' => User::class, 'targetAttribute' => ['userId' => 'id']],  
 ];  
 }  
  
 */\*\*  
 \* {@inheritdoc}  
 \*/* public function attributeLabels()  
 {  
 return [  
 'id' => 'ID',  
 'seatId' => 'ID места',  
 'userId' => 'User ID',  
 'createdAt' => 'Дата создания',  
 'updatedAt' => 'Дата изменения',  
 ];  
 }  
  
 public function toArray(array $fields = [], array $expand = [], $recursive = true)  
 {  
 $res = [  
 'id' => $this->id,  
 'user' => $this->user,  
 'seat' => $this->seat,  
 'voyage' => $this->voyage,  
 ];  
 return [  
*// 'voyageName' => $res['voyage']['name'],* 'voyage' => $res['voyage'],  
*// 'depart' => $res['voyage']['depart'],  
// 'arrive' => $res['voyage']['arrive'],  
// 'departDateTime' => $res['voyage']['departDateTime'],  
// 'arriveDateTime' => $res['voyage']['arriveDateTime'],* 'passenger' => $res['user']['name'],  
 'phone' => $res['user']['phone'],  
 'seatWagon' => $res['seat']['wagonNumber'],  
 'seatNumber' => $res['seat']['seatNumber'],  
 'seatClass' => $res['seat']['class'],  
 'seatPlacement' => $res['seat']['placement'],  
 ];  
 }  
  
 public function behaviors()  
 {  
 return parent::*behaviors*();  
 }  
  
 */\*\*  
 \* Gets query for [[Seat]].  
 \*  
 \* @return \yii\db\ActiveQuery  
 \*/* public function getSeat()  
 {  
 return $this->hasOne(Seat::class, ['id' => 'seatId']);  
 }  
  
 */\*\*  
 \* Gets query for [[User]].  
 \*  
 \* @return \yii\db\ActiveQuery  
 \*/* public function getUser()  
 {  
 return $this->hasOne(User::class, ['id' => 'userId']);  
 }  
  
 */\*\*  
 \* @return \yii\db\ActiveQuery  
 \*/* public function getVoyage()  
 {  
 $seat = new Seat($this->seat);  
 return $seat->getVoyage();  
 }  
  
 */\*\*  
 \* @return \yii\db\ActiveQuery  
 \*/* public function getTrain()  
 {  
 $voyage = new Voyage($this->voyage);  
 return $voyage->getTrain();  
 }  
}

Пример кода контроллера:

<?php  
  
namespace app\modules\v1\controllers;  
use app\modules\v1\models\Seat;  
use app\modules\v1\models\Ticket;  
use yii\db\Query;  
  
class TicketsController extends ApiController {  
  
 public function actionUser\_tickets($userId) {  
 return Ticket::*find*()  
 ->where(['userId' => $userId])  
 ->all();  
 }  
  
 public function actionVoyage\_tickets($voyageId) {  
 return Ticket::*find*()  
 ->where(['voyageId' => $voyageId])  
 ->all();  
 }  
  
 public function actionTickets() {  
 return Ticket::*find*()  
 ->all();  
 }  
  
 public function actionBuy(){  
 $data = \Yii::*$app*->request->getBodyParams();  
 $seat = Seat::*find*()  
 ->where(['id' => $data['seatId']])  
 ->one();  
 $seat->setAttribute('isBusy', '1');  
 $seat->save();  
 $ticket = new Ticket();  
*// $ticket->load($data, '');* $ticket->setAttribute('userId', $data['userId']);  
 $ticket->setAttribute('seatId', $data['seatId']);  
 $ticket->save();  
 return $ticket;  
 }  
}