МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ

УНИВЕРСИТЕТ ИМЕНИ МАШЕРОВА»

Факультет математики и информационных технологий

Кафедра прикладного и системного программирования

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

по дисциплине «веб-технологии»

на тему: «Разработка приложения с использованием RestAPI»

Лепская Дарья Александровна

курс 3 группа 24з

Руководитель:

Никитин Александр Игоревич

Витебск, 2023

**1 Общая часть**

В ходе выполнения проекта необходимо изучить RestAPI и создать небольшой веб-сайт, который будет взаимодействовать с контактами какого-либо человека.

Необходимо создать сервер, который будет поддерживать RestApi и клиента, который сможет отправлять запросы.

**2. Выбор инструментальных средств разработки**

Для реализации веб-приложения были выбраны следующие средства разработки:

* среда разработки «Visual Studio Code»,
* язык гипертекстовой разметки «HTML»,
* язык программирования «JavaScript»,
* программная платформа «Node.js»,
* «RestAPI»,
* фреймворк «Vue.js»

Ниже проанализированы и обоснованы выбранные технологии разработки программного продукта. В этих технологиях максимально эффективно реализовываются поставленные задачи.

**2.1 Microsoft Visual Studio Code**

Microsoft Visual Studio Code – текстовый редактор, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации. [2]

Визуальный редактор кода позволяет работать с IntelliSense — автоматическим дописыванием функций при вводе первых букв. Выполнять отладку - искать и устранять ошибки в написанном коде. Удобно писать код - автоматически заполнять нужную информацию, подсвечивать элементы синтаксиса в зависимости от выбранного языка, расставлять нужные отступы.

**2.2 HTML**

HTML — это язык разметки гипертекстовых документов. Он нужен, чтобы отображать в браузере специальным образом отформатированный документ с множеством вложенных элементов: заголовками, абзацами, списками, гиперссылками, медиаисточниками, расположением изображений, видео и аудио.

Когда пользователь посещает сайт, браузер «подтягивает» файл HTML с данными о структуре и содержании веб-страницы. Функция HTML состоит в выстраивании внешней базы, фундамента, но сам запуск сайта в функционал не входит. HTML только указывает, где должны располагаться элементы, каков их базовый визуал, где брать стили для элементов и скрипты.

**2.3 JavaScript**

JavaScript – это язык программирования, который используют разработчики для создания интерактивных веб-страниц. Функции JavaScript могут улучшить удобство взаимодействия пользователя с веб-сайтом: от обновления ленты новостей в социальных сетях и до отображения анимации и интерактивных карт.

JS поддерживают все популярные браузеры. Во frontend-части сайтов язык используют для создания интерактива (анимаций, всплывающих форм, автозаполнения), так как он связан с HTML и CSS и может ими манипулировать.

В backend-части с языком JavaScript работают на платформе Node.js. С ее помощью, например, разрабатывают серверные веб-приложения и подключают библиотеки. В поисковике Google на JavaScript работает строка автозаполнения, а Netflix, Uber, eBay используют его в своем backend.

**2.4 Программная платформа «Node.js»**

Node.js (Node) — это платформа с открытым исходным кодом для работы с языком JavaScript, построенная на движке Chrome V8. Она позволяет писать серверный код для веб-приложений и динамических веб-страниц, а также программ командной строки. В основе платформы — событийно-управляемая модель с неблокирующими операциями ввода-вывода, что делает ее эффективной и легкой.

Node.js работает на движке V8, транслирующем JavaScript в машинный код. Простыми словами, Node.js — это приложение на C++, которое получает на входе код JavaScript и выполняет его. Чтобы взаимодействовать с устройствами ввода-вывода на компьютере, в платформе предусмотрен собственный интерфейс на C++. Таким образом, платформа превращает специализированный скриптовый язык JavaScript в язык общего назначения, поэтому на Node.js можно писать любые компьютерные программы. Платформа позволяет пользоваться единым языком JavaScript для написания кода и на стороне клиента (Frontend), и на сервере (Backend). Эти возможности Node.js важны для разработки приложений реального времени, которые основаны на событиях.

**2.5 RestAPI**

REST API — это способ взаимодействия сайтов и веб-приложений с сервером. Его также называют RESTful. Термин состоит из двух аббревиатур, которые расшифровываются следующим образом.

*API* (Application Programming Interface) — это код, который позволяет двум приложениям обмениваться данными с сервера. На русском языке его принято называть программным интерфейсом приложения.

*REST* (Representational State Transfer) — это способ создания API с помощью протокола HTTP. На русском его называют «передачей состояния представления».

Технологию REST API применяют везде, где пользователю сайта или веб-приложения нужно предоставить данные с сервера. Например, при нажатии иконки с видео на видеохостинге REST API проводит операции и запускает ролик с сервера в браузере. В настоящее время это самый распространенный способ организации API. Он вытеснил ранее популярные способы SOAP и WSDL.

У RESTful нет единого стандарта работы: его называют «архитектурным стилем» для операций по работе с сервером. Такой подход предложил в 2000 году в своей диссертации программист и исследователь Рой Филдинг, один из создателей протокола HTTP.

У RESTful есть 7 принципов написания кода интерфейсов.

1. *Отделение клиента от сервера (Client-Server).* Клиент — это пользовательский интерфейс сайта или приложения, например, поисковая строка видеохостинга. В REST API код запросов остается на стороне клиента, а код для доступа к данным — на стороне сервера. Это упрощает организацию API, позволяет легко переносить пользовательский интерфейс на другую платформу и дает возможность лучше масштабировать серверное хранение данных.
2. *Отсутствие записи состояния клиента (Stateless).* Сервер не должен хранить информацию о состоянии (проведенных операций) клиента. Каждый запрос от клиента должен содержать только ту информацию, которая нужна для получения данных от сервера.
3. *Кэшируемость (Casheable).* В данных запроса должно быть указано, нужно ли кэшировать данные (сохранять в специальном буфере для частых запросов). Если такое указание есть, клиент получит право обращаться к этому буферу при необходимости.
4. *Единство интерфейса (Uniform Interface).* Все данные должны запрашиваться через один URL-адрес стандартными протоколами, например, HTTP. Это упрощает архитектуру сайта или приложения и делает взаимодействие с сервером понятнее
5. *Многоуровневость системы (Layered System).* В RESTful сервера могут располагаться на разных уровнях, при этом каждый сервер взаимодействует только с ближайшими уровнями и не связан запросами с другими.
6. *Предоставление кода по запросу (Code on Demand).* Серверы могут отправлять клиенту код (например, скрипт для запуска видео). Так общий код приложения или сайта становится сложнее только при необходимости.
7. *Начало от нуля (Starting with the Null Style).* Клиент знает только одну точку входа на сервер. Дальнейшие возможности по взаимодействию обеспечиваются сервером.

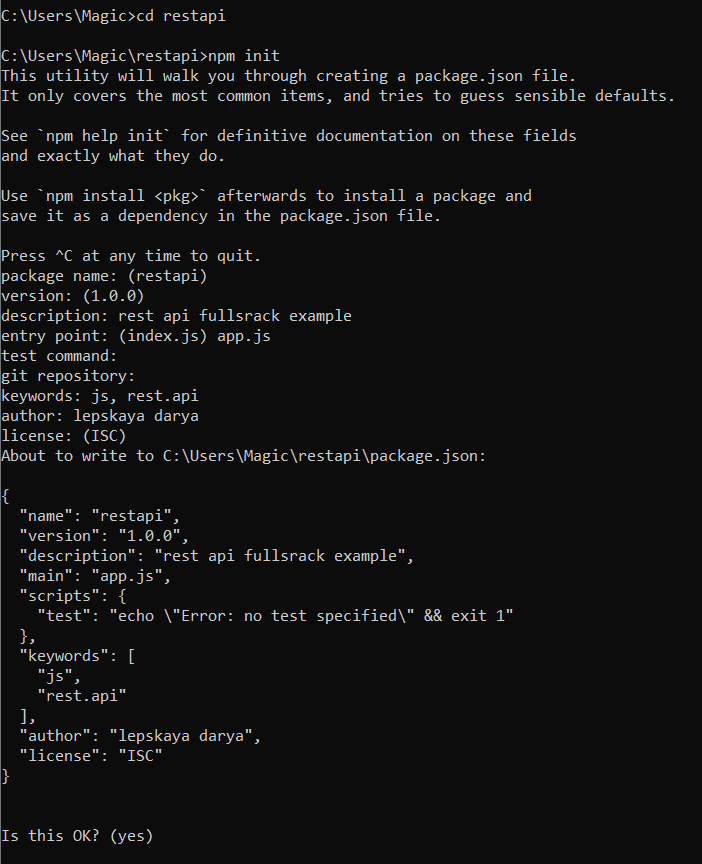
**3 Реализация программного продукта**

Для начала работы подключаем Vue.js с помощью CDN как модуля, с помощью команды:

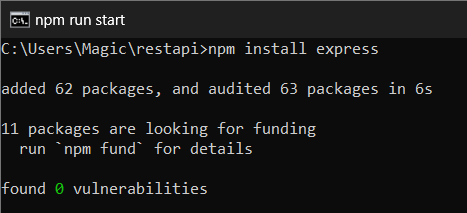
import Vue from 'https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.esm.browser.js'

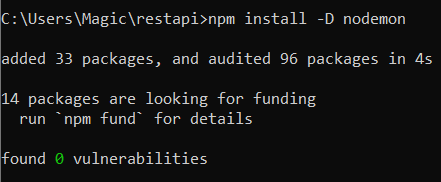
Далее подключаем Node.js, так как мы будем использовать npm-менеджер. Npm (Node Package Manager) – дефолтный пакетный менеджер для JavaScript, работающий на Node.js. Npm помогает взаимодействовать с онлайн-платформами, такими как браузеры и серверы. Эта утилита помогает в установке и удалении пакетов, управлении версиями и зависимостями, необходимыми для запуска проекта.

Далее открываем командную строку и пишем команду: npm init:

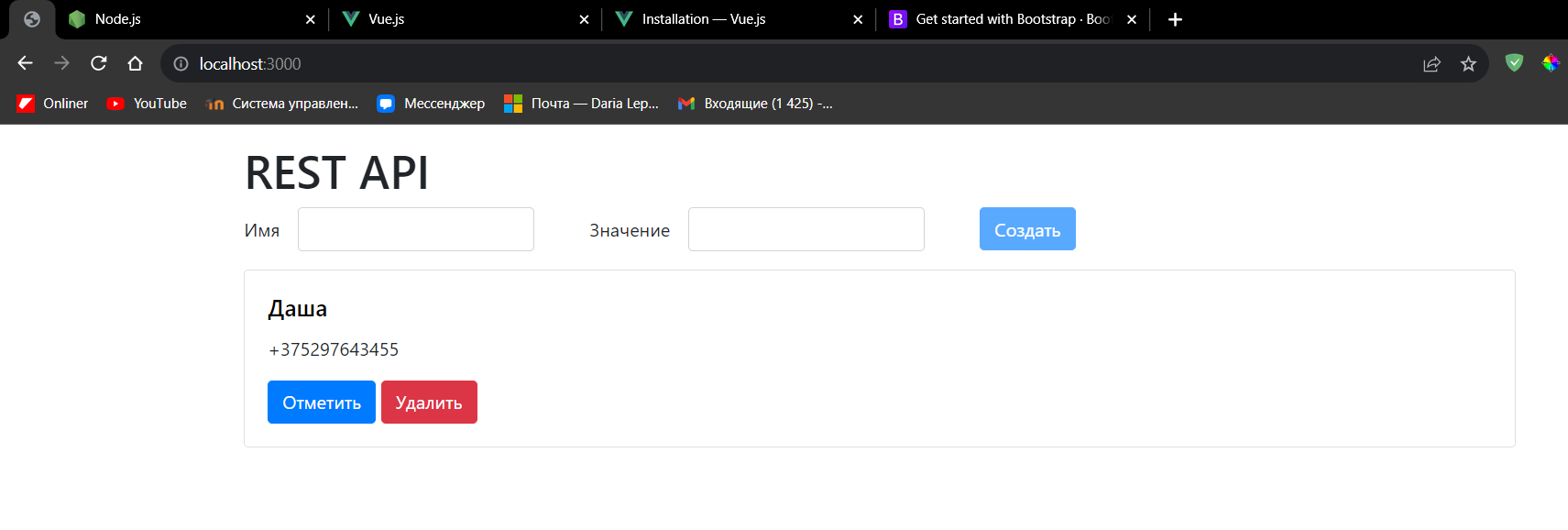


Далее устанавливаем пакет – express и nodemon, для того чтобы комфортно разрабатывать серверную составляющую. Nodemon позволяет перезагружать сервер когда происходят какие-либо изменения.

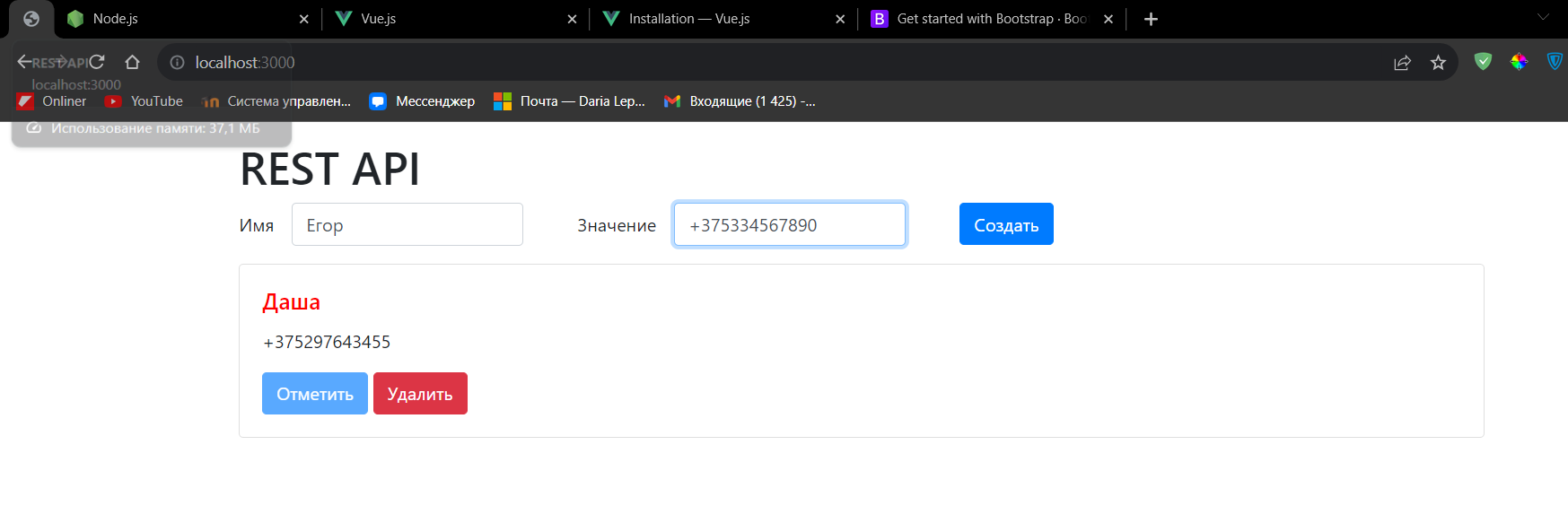




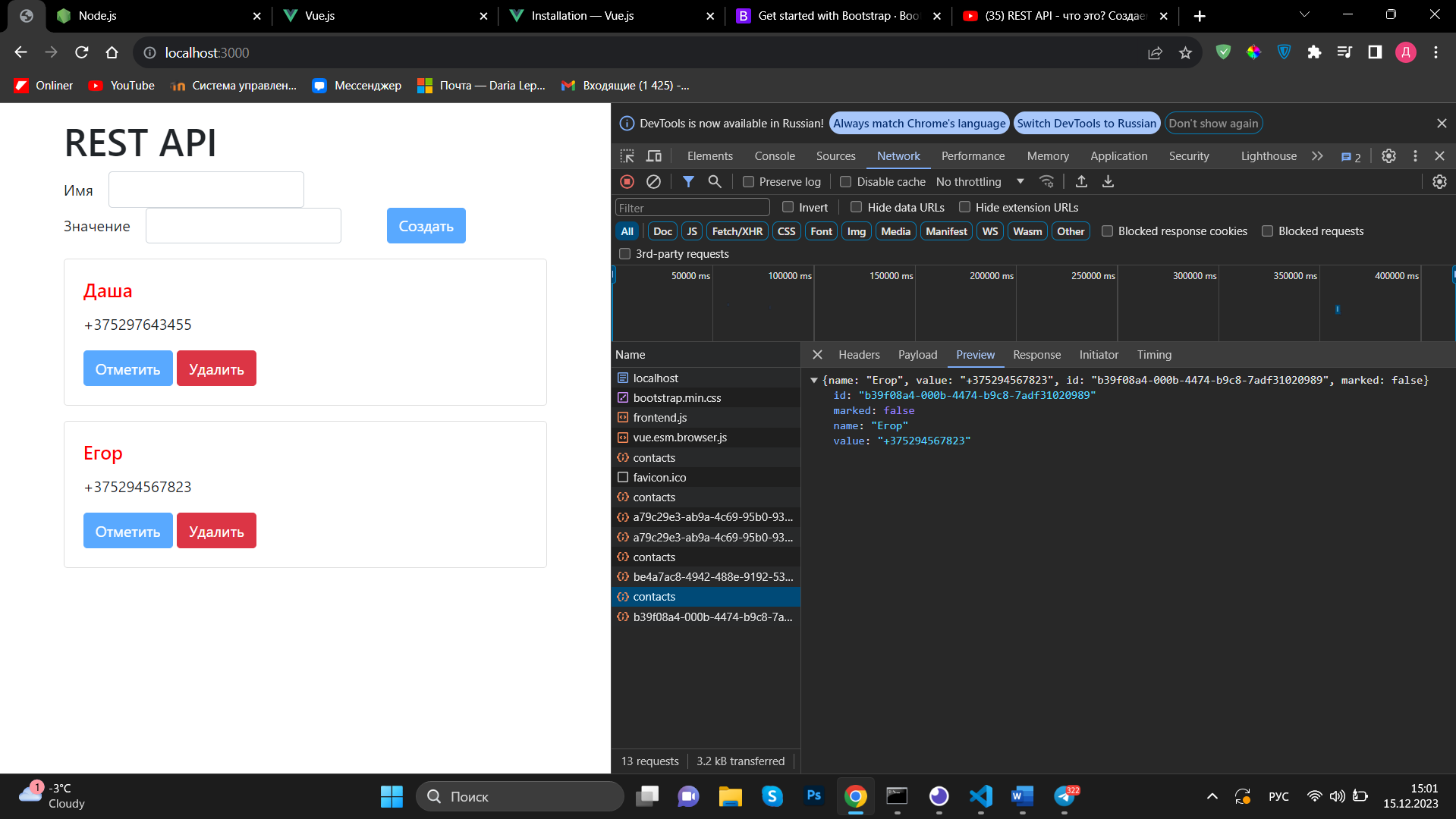
После запуска веб-приложения появляется главная форма. На ней мы видим поля для ввода контакта, и кнопку создания, а так же один контакт, который уже хранится в БД. Его мы можем отметить и удалить.

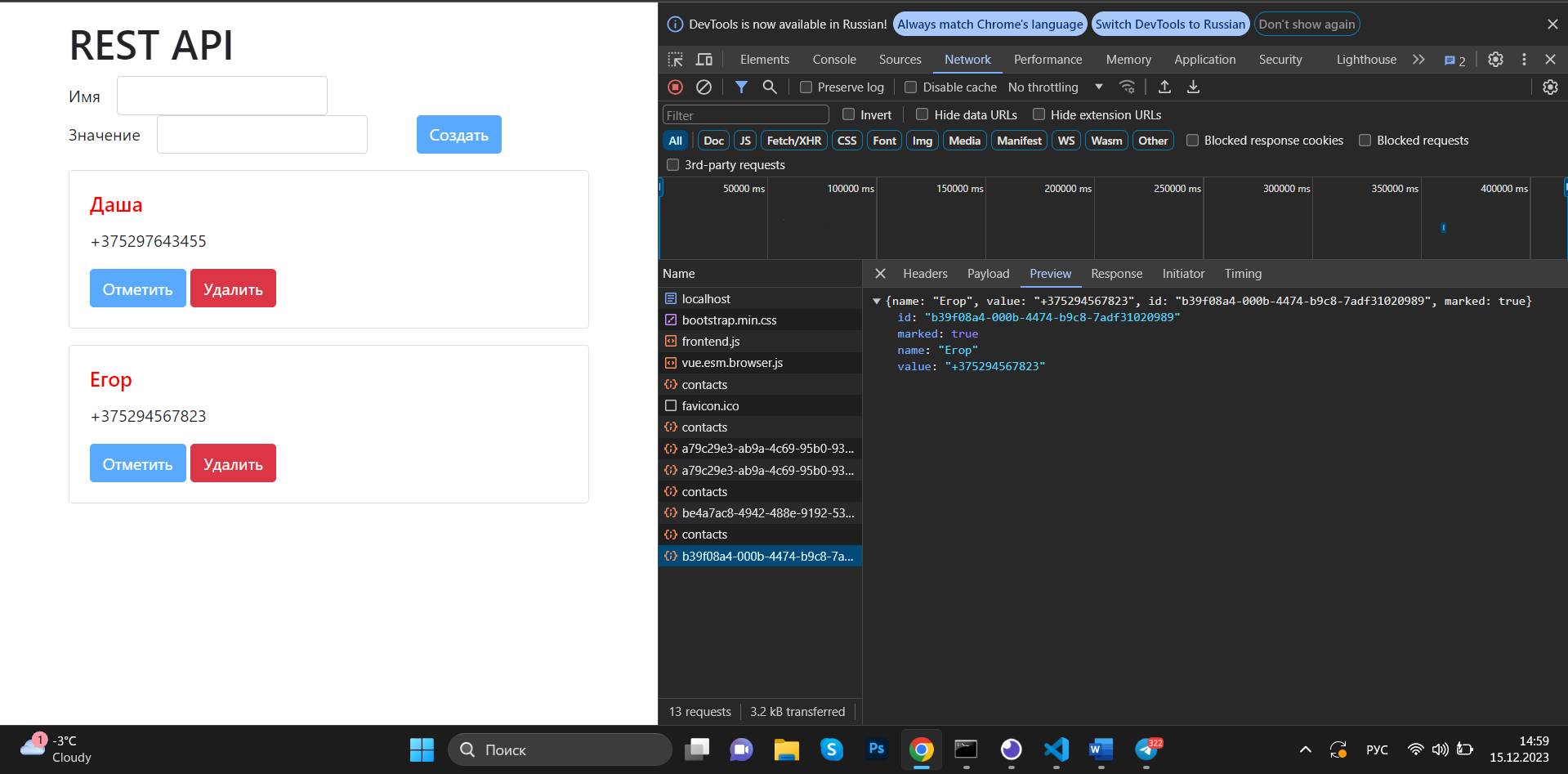


Пока поле «Имя» и «Значение» пустые, кнопка «Создать» не активна. После ввода данных в поля, кнопка активизируется, о чем говорит изменение яркости кнопки.

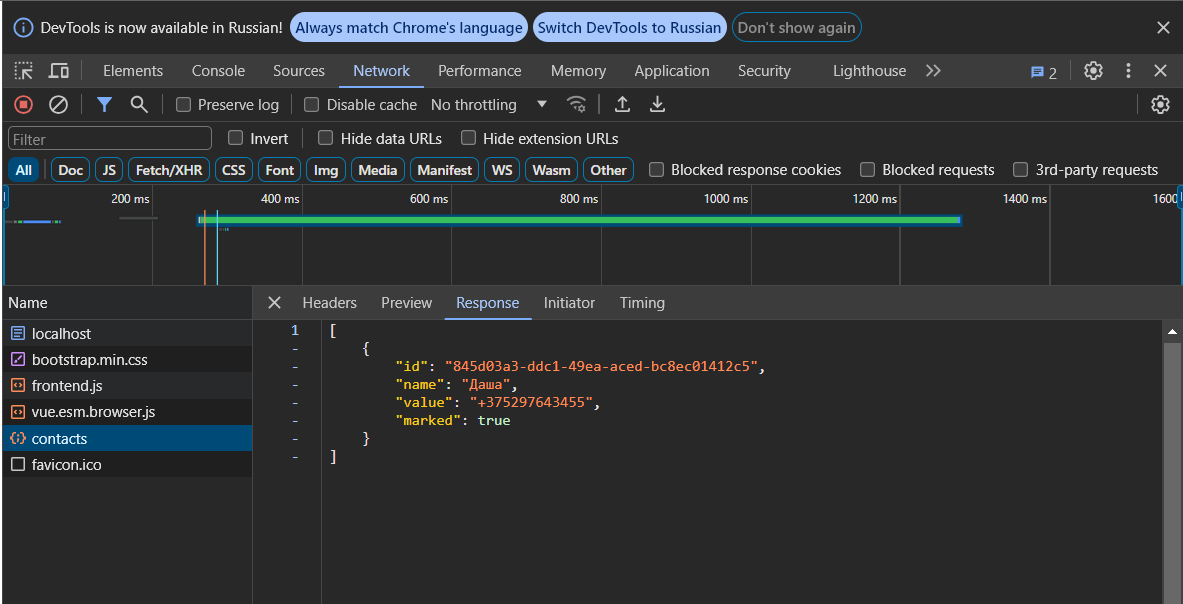


Так же можем добавить контакт и отметить его, после чего имя будет красным, а кнопка «Отметить» станет не активна. Проверим эти действия с помощью консоли. Видим, что после нажатия на кнопку «Отметить» значение marked изменилось с «false» на «true», значит все работает корректно:

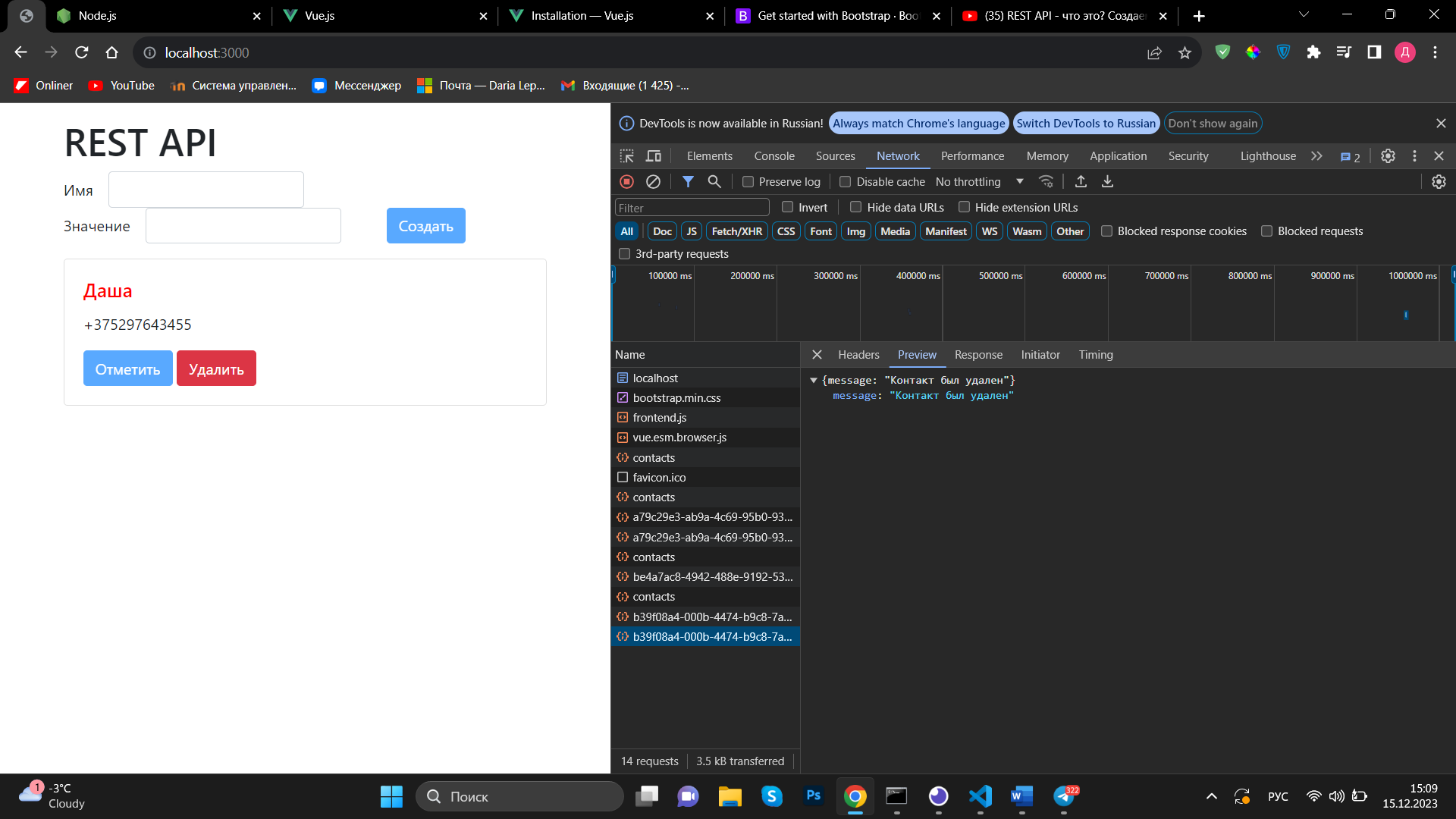




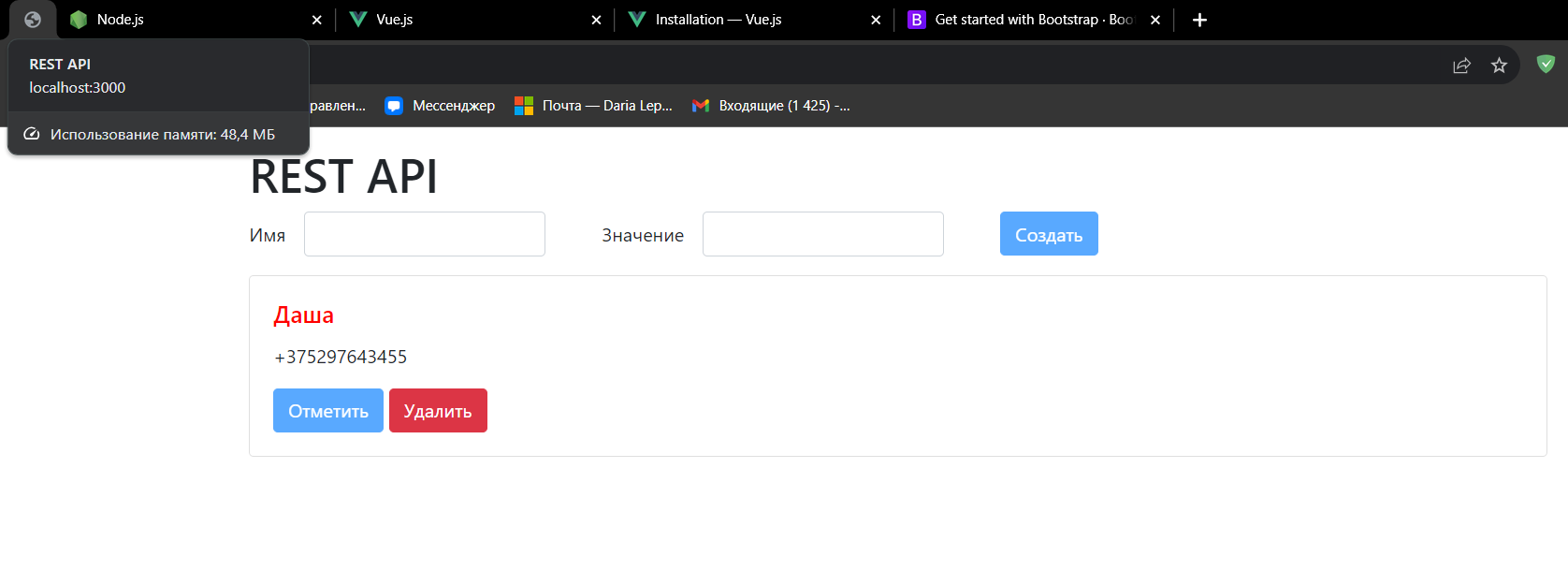
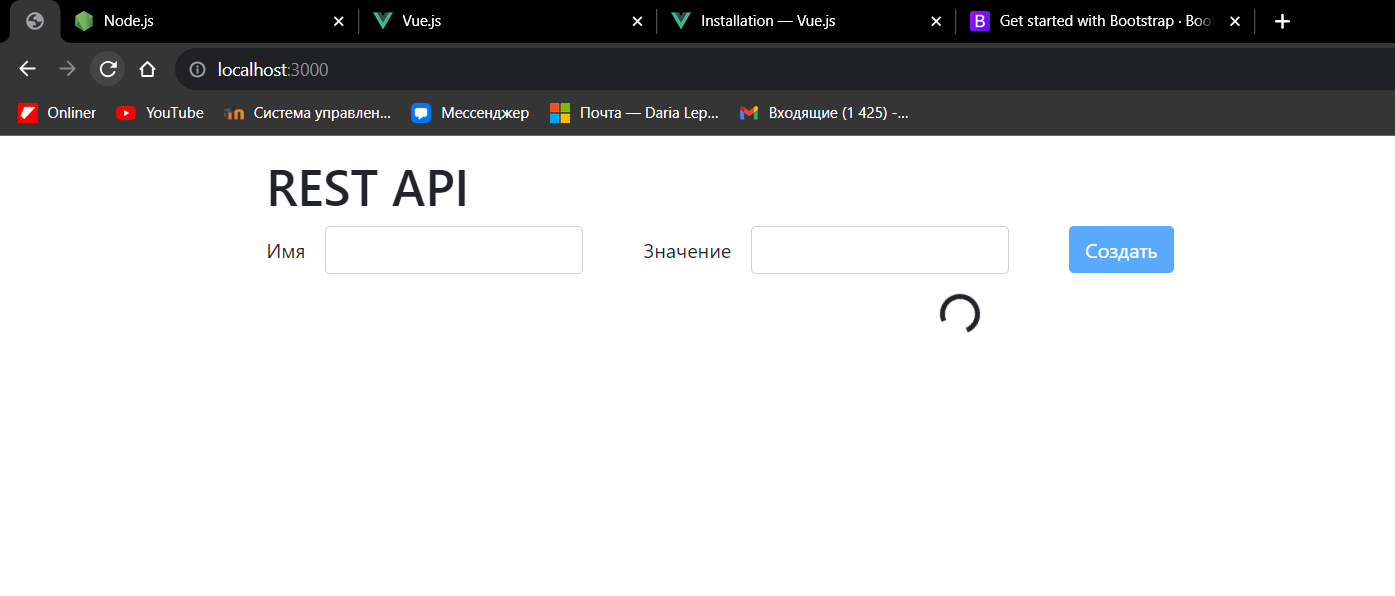
Так же у нас есть объект (база данных), в которой хранится один контакт:



Далее проверим кнопку «Удалить»:



Так же видим, что после обновления старницы все изменения сохраняются:



Основной код программы представлен в приложении А.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ ТЕКСТ ПРОГРАММНОГО СРЕДСТВА

Листинг 1 – Создание всей формы

<form class="form-inline mb-3" @submit.prevent="createContact">

    <div class="form-group mr-5">

      <label for="name" class="mr-3">Имя</label>

      <input type="text" class="form-control" id="name" v-model="form.name">

    </div>

    <div class="form-group mr-5">

      <label for="value" class="mr-3">Значение</label>

      <input type="text" class="form-control" id="value" v-model="form.value">

    </div>

    <button class="btn btn-primary" type="submit" :disabled="!canCreate">Создать</button>

  </form>

  <div v-if="loading">

    <loader />

  </div>

  <div v-else-if="contacts.length">

    <div class="card mb-3" v-for="contact in contacts">

      <div class="card-body">

        <h5 class="card-title" :style="{color: contact.marked ? 'red' : 'black'}">{{contact.name}}</h5>

        <p class="card-text">{{contact.value}}</p>

        <button class="btn btn-primary" @click="markContact(contact.id)" :disabled="contact.marked">Отметить</button>

        <button class="btn btn-danger" @click="removeContact(contact.id)">Удалить</button>

      </div>

    </div>

  </div>

  <p v-else>Контактов пока нет</p>

</div>

Листинг 2 – Создание полей

<div class="form-group mr-5">

      <label for="name" class="mr-3">Имя</label>

      <input type="text" class="form-control" id="name" v-model="form.name">

    </div>

    <div class="form-group mr-5">

      <label for="value" class="mr-3">Значение</label>

      <input type="text" class="form-control" id="value" v-model="form.value">

    </div>

Листинг 3 – Создание кнопок

<button class="btn btn-primary" type="submit" :disabled="!canCreate">Создать</button>

<button class="btn btn-primary" @click="markContact(contact.id)" :disabled="contact.marked">Отметить</button>

<button class="btn btn-danger" @click="removeContact(contact.id)">Удалить</button>

Листинг 4 – Создание объекта Vue, методов добавления, удаления и отметки

new Vue({

  el: '#app',

  data() {

    return {

      loading: false,

      form: {

        name: '',

        value: ''

      },

      contacts: []

    }

  },

  computed: {

    canCreate() {

      return this.form.value.trim() && this.form.name.trim()

    }

  },

  methods: {

    async createContact() {

*const* {...contact} = this.form

*const* newContact = await request('/api/contacts', 'POST', contact)

      this.contacts.push(newContact)

      this.form.name = this.form.value = ''

    },

    async markContact(*id*) {

*const* contact = this.contacts.find(*c* *=>* *c*.id === *id*)

*const* updated = await request(`/api/contacts/${*id*}`, 'PUT', {

        ...contact,

        marked: true

      })

      contact.marked = updated.marked

    },

    async removeContact(*id*) {

      await request(`/api/contacts/${*id*}`, 'DELETE')

      this.contacts = this.contacts.filter(*c* *=>* *c*.id !== *id*)

    }

  },

  async mounted() {

    this.loading = true

    this.contacts = await request('/api/contacts')

    this.loading = false

  }

})

async *function* request(*url*, *method* = 'GET', *data* = null) {

  try {

*const* headers = {}

*let* body

    if (*data*) {

      headers['Content-Type'] = 'application/json'

      body = JSON.stringify(*data*)

    }

*const* response = await fetch(*url*, {

      method,

      headers,

      body

    })

    return await response.json()

  }

  catch (e) {

    console.warn('Error:', e.message)

  }

}

Листинг 5 – Взаимодействие с API (получение данных с помощью метода GET)

// GET

app.get('/api/contacts', (*req*, *res*) *=>* {

  setTimeout(() *=>* {

*res*.status(200).json(CONTACTS)

  }, 1000)

})

Листинг 6 – Метод POST для создания в RestApi

// POST

app.post('/api/contacts', (*req*, *res*) *=>* {

*const* contact = {...*req*.body, id: v4(), marked: false}

  CONTACTS.push(contact)

*res*.status(201).json(contact) //201статус - объект создан

})

Листинг 7 – Метод POST для удаления в RestApi

// DELETE

app.delete('/api/contacts/:id', (*req*, *res*) *=>* {

  CONTACTS = CONTACTS.filter(*c* *=>* *c*.id !== *req*.params.id)

*res*.status(200).json({message: 'Контакт был удален'})

})

Листинг 8 – Метод POST для отметки в RestApi

// PUT

app.put('/api/contacts/:id', (*req*, *res*) *=>* {

*const* idx = CONTACTS.findIndex(*c* *=>* *c*.id === *req*.params.id)

  CONTACTS[idx] = *req*.body

*res*.json(CONTACTS[idx])

})

Листинг 9 – Подключение библиотеки для генерации id

*const* {v4} = require('uuid')