Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ИУ: Информатика и системы управления_____

КАФЕДРА ___ИУ7: Программное обеспечение ЭВМ и информационные технологии_____

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент _____Маклаков Дмитрий Алексеевич_____
фамилия, имя, отчество

Группа _____ИУ7-41Б_____

Тип практики _____Производственная_____

Название предприятия _____МГТУ им. Н. Э. Баумана_____

Студент _____ Маклаков Д. А.
подпись, дата *фамилия, и.о.*

Руководитель практики _____ Куров А. В.
подпись, дата *фамилия, и.о.*

Оценка _____

2020 г.

Оглавление

Цель и задачи практики

Цель практики – апробация знаний и навыков в проектировании, прототипировании и кодировании информационных систем, полученных в ходе обучения, при реализации прикладных задач.

К задачам практики относятся применение знаний для построения информационных систем; использование аналитического подхода для выбора наиболее оптимальных технологий и инструментов программной реализации индивидуального продукта; воплощение в коде, макетах и прототипах программного продукта, соответствующего индивидуальному заданию.

Индивидуальное задание

Практика проходила в виде командной работы над проектом. Задача, поставленная перед командой, – разработка web-приложения, позволяющего находить потенциальных единомышленников в МГТУ, при помощи анализа полей интересов профиля социальной сети «ВКонтакте».

Входные данные: учётная запись пользователя в социальной сети «ВКонтакте».

Выходные данные: список наиболее подходящих по интересам людей для авторизовавшегося пользователя.

В рамках этой задачи мне необходимо было разработать клиентскую часть web-приложения: проанализировать дизайн приложений схожей тематики, изучить последние тренды в web-дизайне, создать макет будущего дизайна. Далее, используя один из фреймворков для конструирования сайта и языки программирования HTML (в том числе HTML5), CSS (в том числе CSS3), реализовать некоторые страницы сайта.

Организовать взаимодействие с серверной частью приложения, а также отображение полученных данных, в том числе, вывести на странице с результатами поиска графовое представление результатов.

Содержание работ, проведенных за время практики

Работу, проведённую во время практики, можно разбить на несколько этапов, среди которых: анализ задачи; выбор инструментов разработки (как для проектирования приложения, так и для программной реализации); построение архитектуры приложения; прототипирование приложения; его программная реализация.

1. Анализ задачи

В современных условиях у людей всё меньше времени остаётся на общение с новыми людьми, поэтому все большей популярностью пользуются различные сервисы для поиска новых знакомств. Анализ рынка показал, что большинство современных приложений, таких как Tinder, не проводят автоматический анализ информации о пользователе, т. е. необходимо самостоятельно заполнять значительное количество информации о себе; также многим решениям недостает наглядной визуализации и удобного интерфейса.

Как следствие, было необходимо разработать простое и удобное приложение, позволяющее найти единомышленников, друзей по интересам, используя информацию из социальных сетей для автоматического сбора информации о пользователях.

2. Выбор инструментов разработки

Путём анализа современных web-приложений, было принято решение о следовании SPA-методологии (Single Page Application – одностраничное web-приложение). Данную методику целесообразно использовать, поскольку она позволяет сэкономить время и трафик пользователя из-за отсутствия необходимости полностью загружать каждую страницу сайта (возможно выделение постоянных блоков-компонентов на странице), имеется необходимость в насыщенном пользовательском интерфейсе, кроме того, одностраничные приложения имеют лучшую кроссплатформенность и адаптивность.

Для создания приложения использовались языки HTML (в том числе и его современная версия HTML5), CSS (в частности, CSS3), а также язык JavaScript и его расширение TypeScript [1], позволяющее работать со статической типизацией, использовались для создания динамики на сайте, работы с серверной частью приложения.

Для создания web-приложения использовался фреймворк Angular; выбор обусловлен современностью данной технологии, простотой и большим количеством материалов в сети, позволяющих быстро освоить и начать использовать фреймворк.

Также использовались библиотеки RxJs для создания динамического приложения и d3 для работы с графом и вывода результата поиска потенциальных единомышленников в виде связного графа. Библиотека d3 была выбрана, поскольку она обширным функционалом и возможностями, которые предоставляет данная библиотека: стилизацией объектов, возможностью обрабатывать события (к примеру, нажатие на объект), наличием большого количества обучающего материала и примеров в сети.

При проектировании пользовательского интерфейса использовался инструмент Figma, функционал которого, среди прочего, позволяет легко и удобно прототипировать и макетировать интерфейсы, в частности, возможны группировка, выравнивание объектов, возможность экспорта картинок в различные форматы для дальнейшего использования в вёрстке. Один из плюсов приложения Figma – возможность самостоятельно настраивать, а в дальнейшем копировать стили различных элементов как в виде данных (к примеру, размера текста и шрифта), так и в виде готового CSS-кода, который можно сразу вставить в файл стилей и использовать при вёрстке. Экспортировать самостоятельно нарисованные полигоны можно сразу в векторном формате svg, благодаря чему не теряется качество при различных расширениях.

3. Построение архитектуры приложения

В архитектуре приложения, представленной на рисунке 1, можно выделить клиентскую и серверную части, контроллер, осуществляющий связь между различными частями проекта, а также базу данных mongodb. На серверной части осуществляется работа с данными, отправленными с пользовательской части, работа с базой данных.

В качестве индивидуального задания мне необходимо было реализовать клиентскую часть приложения (App на рисунке 1) и организовать отправку запросов и данных на контроллер, а также приём запросов и данных от него.

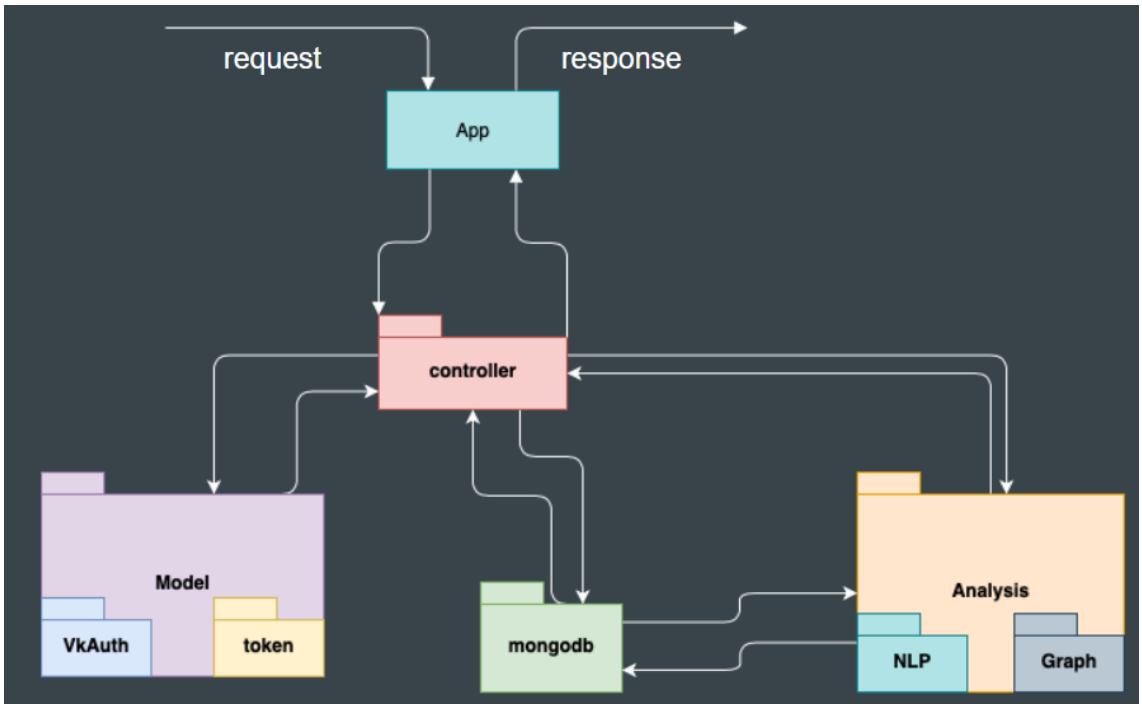

Рисунок 1. Архитектура приложения

4. Прототипирование приложения

Для разработки удобного пользовательского интерфейса на первом этапе был произведён анализ приложений подобной тематики с целью обнаружения интересных визуальных приёмов, изучения возможного поведения пользователей на страницах, кроме того, был изучен материал по современным трендам в web-дизайне. Для поиска интересных дизайнерских решений дополнительно использовалось приложение Pinterest, позволяющее находить

изображения интересующей тематики, в нашем случае – современных дизайнов web-приложений.

После выбора некоторых решений началось создание прототипов в приложении Figma [2]. Было создано несколько различных макетов главной страницы, проведён опрос потенциальных пользователей, после чего было выбрано наиболее популярное решение с учётом недочётов, указанных сторонними наблюдателями. На рисунках 2 и 3 представлены макеты, созданные в приложении Figma.

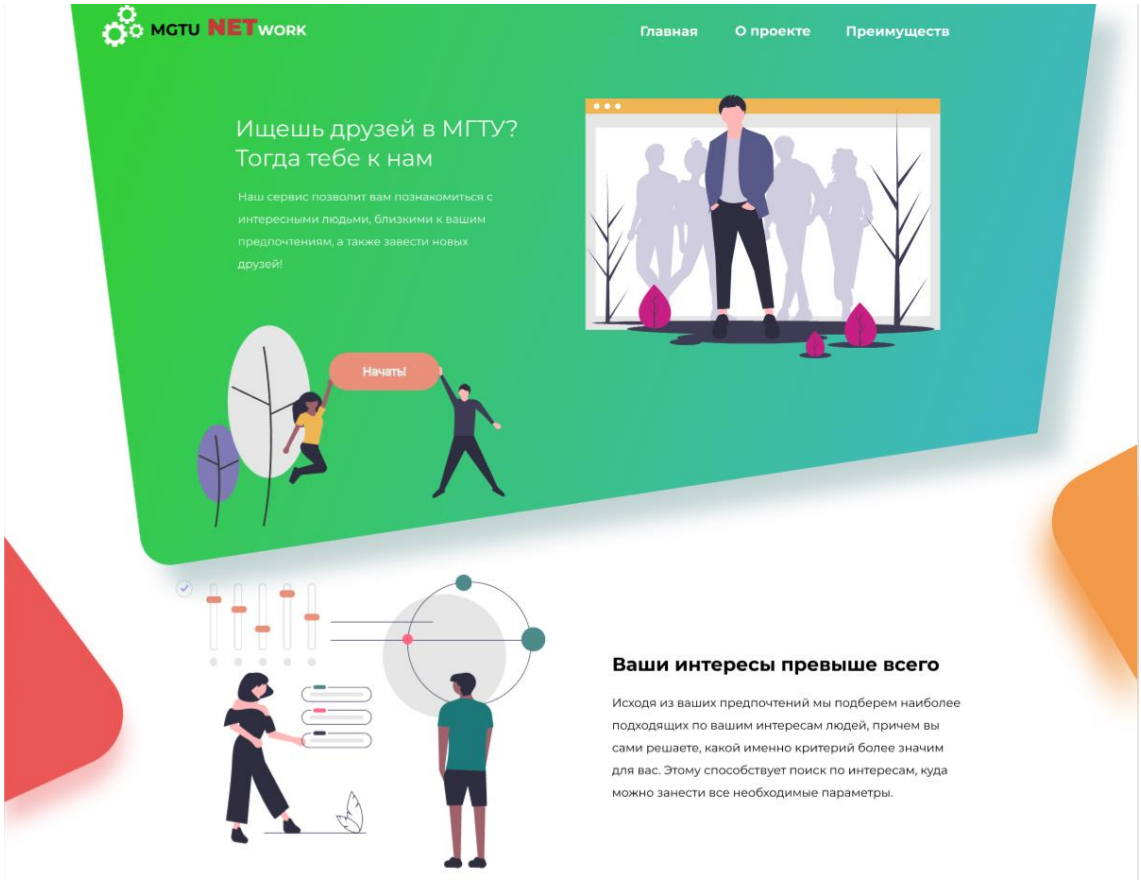

Рисунок 2. Макет дизайна главной страницы в программе Figma

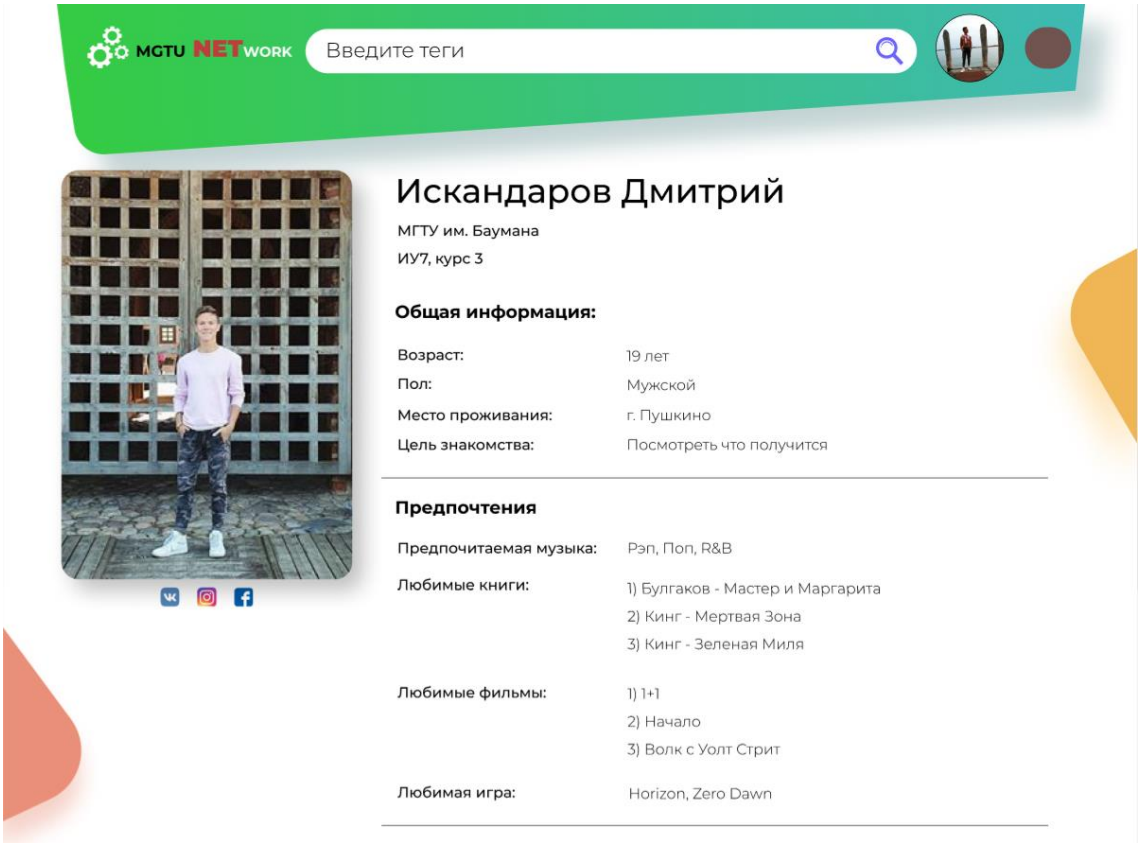

Рисунок 3. Макет дизайна профиля в программе Figma

Как видно из представленных макетов, была выбрана светлая цветовая гамма (белый фон), в качестве ярких акцентов были выбраны красный, жёлтый и зелёный цвета, использованные в различных элементах оформления сайта, например, в шапке и футере, в боковых полигонах.

5. Программная реализация

После создания макетов в приложении Figma началась их реализация в виде кода с использованием HTML, CSS на основе фреймворка Angular [3]. Каждая страница была разбита на три блока: верхняя часть (шапка), средняя (основное содержимое), нижняя часть (футер, копирайт). Это было сделано для разбиения страницы на отдельные компоненты в Angular с целью создания на сайте удобной маршрутизации.

Основной объект в Angular – компонент [4] – отдельный блок кода, который можно вставить на страницу, в том числе и несколько раз. В общем

случае, компонент лежит в одноимённой папке и состоит из трёх файлов (см. рисунок 3):

- файла разметки с расширением .html;
- файла стилей с расширением .css (или .scss);
- файла скриптов с расширением .ts (от TypeScript).

Название файлов в папке компонента имеет следующую структуру: «имя.component.расширение».

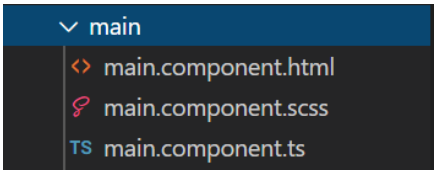

Рисунок 4. Структура компонента main

Основная страница (файл index.html) включает в себя компонент <app>, в который уже можно вставлять другие компоненты, используя соответствующий тег: <имя></имя>.

Одна из возможностей фреймворка Angular – возможность создания маршрутизации [5], то есть, подгрузки различных компонентов в зависимости от адреса. Для этого необходимо импортировать классы Routes, RouterModule из библиотеки ‘@angular/router’, создать файл app-routing.module.ts и прописать тег <router-outlet></router-outlet> в нужном компоненте (в нашей случае – в компоненте app). Реализация маршрутизации в рамках проекта представлена в листинге 1.

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { MainComponent } from './main/main.component';
import { NotFoundComponent } from './not-found/not-found.component';
import { AboutComponent } from './about/about.component';
import { AuthComponent } from './auth/auth.component';
import { SearchComponent } from './search/search.component';
import { UserComponent } from './user/user.component';
import { HeadSmallComponent } from './head-small/head-small.component';
import { GraphComponent } from './graph/graph.component';
import { ExitComponent } from './exit/exit.component';


const routes: Routes = [
  { path: 'vk_get_auth', component: AuthComponent },
  { path: '', component: MainComponent },
  { path: 'about', component: AboutComponent },
  { path: 'search', component: SearchComponent },
  { path: 'main', redirectTo: '', pathMatch: 'full' },
  { path: 'user/:token', component: UserComponent },
  { path: 'user', redirectTo: 'user/' + localStorage.getItem('token'), pathMatch: 'full' },
  { path: 'exit', component: ExitComponent },
  { path: '**', component: NotFoundComponent, pathMatch: 'full' }
]

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Листинг 1. Реализация маршрутизации, файл app-routing.module.ts

В константе routes прописываются обрабатываемые пути, далее указывается, какой из компонентов необходимо включить при совпадении пути с указанным. Путь ‘**’ соответствует случаю, когда путь не был найден среди указанных выше (компонент «Страница не найдена»); pathMatch: ‘full’ – указание на проверку полного соответствия пути; redirectTo – адрес для перенаправления при необходимости.

```
<div class = 'main'>
  <div class = 'content'>
     <router-outlet></router-outlet>
     <bc-footer></bc-footer>
  </div>
  <bc-side-figures></bc-side-figures>
</div>
```

Листинг 2. Файл app.component.html

В листинге 2 показано применение тега <router-outlet>, указывающего на использование маршрутизации: в место, где указан тег, будет подставлен соответствующий данному пути компонент из app-router.module.ts. Преимущество использования маршрутизации – возможность динамически подгружать нужный модуль, а также экономить время и ресурсы пользователя, не загружая повторно модули, нужные на каждой странице. В нашем случае (см. листинг 2) – это компоненты <bc-footer> (нижняя часть страницы, копирайт) и <bc-side-figures> (боковые полигоны слева и справа от содержимого, см. рисунок 1). Приставка bc- может быть установлена создателем проекта для более явного отличия от встроенных тегов HTML: к примеру, у нас есть свой компонент <bc-footer>, в который включён тег HTML5 <footer> – без приставки были бы проблемы с распознаванием. На рисунке 5 и в листинге 3 представлена реализация шапки сайта.

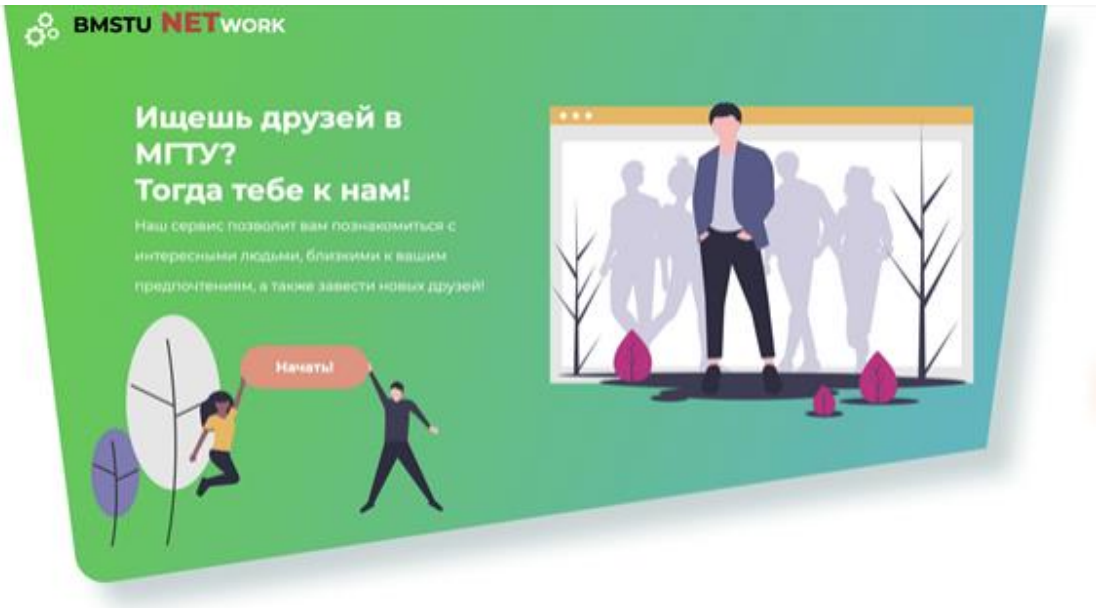

Рисунок 5. Реализация шапки по макету

```
<header>
    <img src = '../../images/head.png' alt = 'header'>
    <div class = 'head_content'>
        <div class = 'head_up'>
            <div class = 'head_left'>
                <div class = 'head_logo'>
                    <img  src  =  '../../images/logo.png'  alt  =
'logo'>
                </div>
                <div class = 'head_title'>
                    <span class = '_1'>BMSTU</span><span class =
'_2'> NET</span><span class = '_3'>WORK</span>
                </div>
            </div>
        </div>
        <div class = 'head_last'>
            <div class = 'left'>
                <div class = 'text'>
                    <span   class   =   'title'>Ищешь   друзей   в
МГТУ?<br>Тогда тебе к нам!<br></span>
                    Наш сервис позволит вам познакомиться с
интересными людьми, близкими к вашим предпочтениям, а также
завести новых друзей!
                </div>
                <div class = 'button'>
                    <img src = '../../images/head-left.svg'>
                    <a     class     =     'start'     href     =
'https://oauth.vk.com/authorize?client_id=7543477&display=page&r
edirect_uri=http://127.0.0.1:4200/vk_get_auth&scope=friends&resp
onse_type=code&v=5.120' target = '_self'>Начать!</a>
                </div>
            </div>
            <div class = 'right'>
                <img src = '../../images/head-right.svg'>
            </div>
            <div class = "clear"></div>
        </div>
    </div>
</header>
```

Листинг 3. Реализация шапки сайта, файл head.component.html

В приведённом выше листинге шапки сайта (.html) видно, что при нажатии пользователем на кнопку авторизации происходит переход на страницу авторизации социальной сети ВКонтакте. В листинге 4 представлены стили, применяемые к компоненту шапки.

```
header {
    margin: 0 auto;
    padding: 0;
    position: relative;
}

.head_content {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
}

.head_left {
    margin-left: 20px;
    float: left;
    height: 100%;
    display: table;
}

.head_logo {
    margin: auto;
    height: 100%;
    display: table-cell;
    vertical-align: middle;
}

.head_last {
    font-family: 'Montserrat', sans-serif;
    color: #ffffff;
    margin: 0 100px;
}

.head_last .left .button .start {
    position: absolute;
    top: 35px;
    left: 170px;
    background-color: #ea907a;
    border-radius: 25px;
    padding: 15px 40px;
    border: none;
    color: #ffffff;
    font-weight: 700;
}
```

Листинг 4. Стилевое оформление шапки сайта (часть): head.component.scss

Используя CSS, а также его современную версию CSS3, блоки страницы были оформлены согласно макетам, представленным выше. Цвета, а также

шрифты, можно удобно копировать из программы Figma, в которой был создан макет сайта.

Для работы с серверной частью приложения и отправки туда запросов была использована технология ApiService. После импорта HttpClient из библиотеки ‘@angular/common/http’ появляется возможность отправки на различные адреса POST и GET запросов с данными, а также возврат этих данных в месте вызова соответствующих функций. Был создан файл api.service.ts (см. листинг 5), в котором были реализованы функции для взаимодействия с серверной частью приложения: получения уникального токена пользователя после авторизации в социальной сети ВКонтакте, получение данных о пользователе по токену из базы данных, выход пользователя из профиля, получение графа возможнвх единомышленников по указанным интересам. Имеются как POST-запросы, в которых информация передаётся в виде объекта типа object, так и GET-запросы, в которых информация передаётся через параметры адресной строки. Отдельно стоит обратить внимание на константу environment.apiBaseUrl, в которой был заранее указан адрес серверной части приложения. Это позволяет не указывать его каждый раз вручную, а также быстро корректировать его при необходимости.

Данные функции вызываются после авторизации, при переходе на страницу пользователя, при переходе на страницу поиска (или нажатие на ней соответствующей кнопки), а также при нажатии кнопки «Выход».

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from '../environments/environment';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(
    private readonly http: HttpClient
  ) { }

  async exitByToken(token: string) {
    const url =
`${environment.apiBaseUrl}/api/users/exit/${token}`;
    const result = await this.http.post(url, {}).toPromise();
    return result;
  }

  async getTokenByCode(code: string) {
    const url = `${environment.apiBaseUrl}/api/login/${code}`;
    const token = await this.http.get(url).toPromise();
    return token;
  }

  async getGraphByDates(depth: number, graph: object) {
    const token = localStorage.getItem('token');
    const url =
`${environment.apiBaseUrl}/api/graph/${depth}/${token}`;
    const result = await this.http.post(url, graph).toPromise();
    return result;
  }

  async loadUserByToken(token: string) {
    const url = `${environment.apiBaseUrl}/api/users/${token}`;
    const user = await this.http.get(url).toPromise();
    return user;
  }
}
```

Листинг 5. Реализация api.service.ts

Наибольший интерес представляет из себя работа с графом и поиском потенциальных единомышленников, поскольку необходимо было встроить библиотеку D3 в приложение Angular, обработать стили и выводимую

информацию, реализовать обработку нажатий на вершины графа (см. рисунок 6, листинг 6).

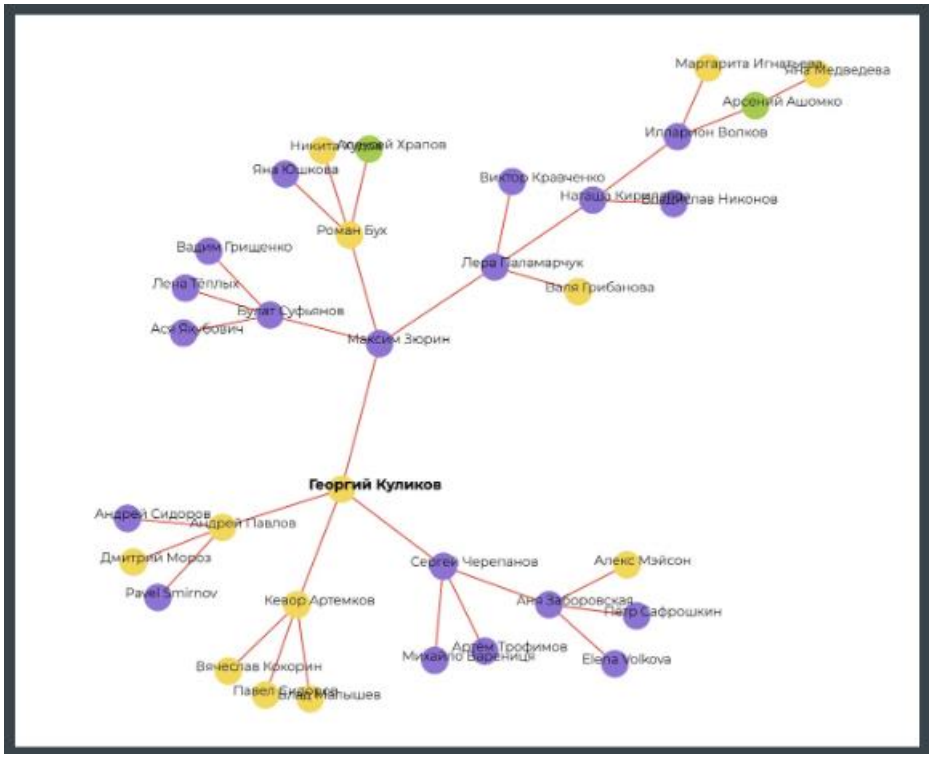

Рисунок 6. Пример графа

Авторизовавшийся пользователь, для которого осуществляется поиск, выделен жирным шрифтом, имеет потомков, на которые можно нажимать либо для сворачивания / разворачивания их потомков, либо для перехода на профиль пользователя. Каждый из узлов графа имеет цвет заливки, соответствующий совпадающему интересу. Сам граф представляет собой svg-холст, на который накладываются рёбра (прямые) и узлы (круги). Была также произведена интеграция стилей svg с классами scss.

Сам граф был помещён в компонент SearchComponent, работа с библиотекой d3 [6] производится в файле search.component.ts. Помимо вставки гарфа, в файле search.component.html (см. листинг 6) присутствует вывод пользователю кнопок, отвечающих за интересы. Данные кнопки позволяют увеличить вес соответствующего поля с точки зрения графа. Также присутствует тег вставки компонента шапки <bc-head></bc-head>.

```
<bc-head></bc-head >
<div class="coolPic">
    <img src="../../images/coolPic.svg" width=180px>
</div>

<div class="filters">
    <div class="tags">
          <div *ngFor="let it of tagArray; let i = index">
            <button [name]="clickArray[i]"
(click)="clickArray[i][0] = !clickArray[i][0]">{{it}}</button>
        </div>
    </div>
     <div class="question__mark">
            <div class="mark">
                <a href="#markPressed">
                    <img src="../../images/question.png" alt=""
title="нажмите на тег, чтобы его выбрать. эти интересы станут
более приоритетными" width="20px">
                </a>

                <div id="markPressed" class="explaination">
                    <div>
                        <a href="#close" title="Закрыть"
class="close2">X</a>
                        <p>Нажмите на тег, чтобы его выбрать.
Эти интересы станут более приоритетными</p>
                    </div>
                </div>
            </div>
        </div>
    <div class="search__btn">
        <button
(click)="sendFilters(clickArray)">Применить</button>
    </div>
</div>
<svg width="960" height="600"></svg>
```

Листинг 6. Код файла search.component.html

В файле search.component.ts (см. листинг 7) производится работа по получению графа от серверной стороны, трансформирование в svg-элементы при помощи библиотеки d3, вывод графа на страницу.

```
import { Component, OnInit } from '@angular/core';
import { ApiService } from '../api.service';

import * as d3 from 'd3';

@Component({
  selector: 'bc-search',
  templateUrl: './search.component.html',
  styleUrls: ['./search.component.scss']
})
export class SearchComponent implements OnInit {
  public root;

  sendFilters(jsonFilters: object) {
    this.api.getGraphBySmth(0, jsonFilters).then(result => {
      this.root = result['graph'];
      this.update();
    });
  }

  constructor(private readonly api: ApiService) {}

    const color = d3.scaleOrdinal(d3.schemeCategory10);

    const simulation = d3.forceSimulation()
      .force('link', d3.forceLink().id((d: any) => d.id))
      .force('charge', d3.forceManyBody().strength(-20))
      .force('center', d3.forceCenter(width / 2, height / 2));
      var nodes = [];
      var links = [];


        var id = 0;
        function recurse(node) {
          if (!node.id) node.id = ++id;
          nodes.push(node);
          if (node.children) {
            node.children.forEach(function(item, i, arr) {
              links.push({"source":node, "target":item["child"],
"distance":item["edge_length"]});
              recurse(item['child']);
            });
          }
        }

        recurse(this.root);

        const graph: Graph = <Graph>{ nodes, links };


        const link = svg.append('g')
          .attr('class', 'link')
```

```
          .selectAll('line')
          .data(graph.links)
          .enter()
          .append('line')
          .attr('stroke', "#f00");

        const node = svg.append('g')
          .attr('class', 'node')
          .selectAll('circle')
          .data(graph.nodes)
          .enter()
          .append('circle')
          .attr('r', 10)
          .attr('fill', function(d: any) { return colors[d.tag];
})
          .attr('stroke', function(d: any) { return
colors[d.tag]; })
            .on("click", this.click)
          .attr('cursor', 'pointer')

        const label = svg.append('g')
          .selectAll('text')
          .data(graph.nodes)
          .enter().append('text')
          .text(function (node:any)  {return node.first_name +
'\r\n' + node.last_name; })
          .attr('font-size', (node:any) => {
            if (node['last_name'] == this.root['last_name']) {
return '11px'; }
            else { return '10px'; }
          })
          .attr('text-align', 'center')
          .on("click", this.click)
          .attr('cursor', 'pointer')
          .attr('user-select', 'none')
            .attr('text-anchor', 'middle')
          .attr('font-weight', (node) => {
            if (node['last_name'] == this.root['last_name']) {
return '600'; }
            return '300';
          })

        svg.selectAll('circle').call(d3.drag()
          .on('start', dragstarted)
          .on('drag', dragged)
          .on('end', dragended)
        );

        simulation
          .nodes(graph.nodes)
          .on('tick', ticked);
```

```
        simulation.force<d3.ForceLink<any, any>>('link')
          .links(graph.links)
          .distance(function(link, i, links) { return
link.distance * 5; });

        function ticked() {
          link
            .attr('x1', function(d: any) { return d.source.x; })
            .attr('y1', function(d: any) { return d.source.y; })
            .attr('x2', function(d: any) { return d.target.x; })
            .attr('y2', function(d: any) { return d.target.y; });

          node
            .attr('cx', function(d: any) { return d.x; })
            .attr('cy', function(d: any) { return d.y; });
          label
            .attr('x', function(d: any) { return d.x; })
            .attr('y', function(d: any) { return d.y; });
        }

    function dragstarted(d) {
      if (!d3.event.active) {
simulation.alphaTarget(0.3).restart(); }
      d.fx = d.x;
      d.fy = d.y;
    }

    function dragged(d) {
      d.fx = d3.event.x;
      d.fy = d3.event.y;
    }

    function dragended(d) {
      if (!d3.event.active) { simulation.alphaTarget(0); }
      d.fx = null;
      d.fy = null;
    }
  }

  ngOnInit() {
    this.api.getGraphBySmth(0, {}).then(result => {
      this.root = result['graph'];

       function get_start(node) {
          if (node.children) {
               node._children = node.children
               node.children = null;
               node._children.forEach(function(item, i, arr) {
                    get_start(item['child']);
            });
          }
        }
```

```
          get_start(this.root);
          this.root.children = this.root._children;
          this.root._children = null;
          this.root.children.forEach(function(item, i, arr) {
               item['child'].children = item['child']._children;
               item['child']._children = null;
          });
      this.update();
    });
  }

  click = (d) => {
    if (d3.event.defaultPrevented) return; // ignore drag
    if (d3.event.shiftPressed) {
          if (d.children) {
            d._children = d.children;
            d.children = null;
          } else {
            d.children = d._children;
            d._children = null;
          }
     }
     else {
          if (d.token) {
               window.open(`/user/' + d.token, '_blank');
          }
          else {
               window.open('http://' + d.vk_link, '_blank');
          }
     }
     this.update();
     }
}
```

Листинг 7. Файл search.component.ts

Таким образом, были реализованы основные страницы web-приложения, а также организованы маршрутизация и взаимодействие с серверной стороной.

Результат работы

В качестве результата, было спроектировано web-приложение, которое анализирует интересы пользователей, имеющихся в базе данных, и ищет единомышленников для различных целей. Для привлечения пользователей была разработана яркая и минималистичная концепция дизайна. На главной странице потенциальному пользователю описываются преимущества сервиса, а также имеется кнопка авторизации. Страница профиля позволяет просматривать информацию о себе или других пользователях, имеется также функция редактирования информации о себе. На странице поиска пользователь может увидеть в графовом представлении потенциальных единомышленников, при этом возможна расстановка приоритетов, а также переход к страницам пользователей для просмотра подробной информации.

Выводы по результатам практики

В результате прохождения производственной практики были применены знания и умения, полученные во время обучения; получен опыт командной работы над крупным проектом; опыт работы в команде с другими программистами и в работе с чужим кодом. Были получены знания по работе с фреймворком Angular, программами Figma, Visual Code, языками программирования HTML (HTML5), CSS (CSS3), JavaScript, TypeScript, библиотеками RxJs, d3. В качестве программного продукта были созданы прототип, макет и программная реализация проекта для поиска потенциальных единомышленников.

Список использованных источников

1. Основы TypeScript, необходимые для разработки Angular-приложений, [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/ruvds/blog/344502/, свободный – (дата обращения: 17.07.2020)
2. Официальный сайт Figma, документация. [Электронный ресурс] – Режим доступа: https://help.figma.com/hc/en-us, свободный – (дата обращения: 10.07.2020)
3. Официальный сайт Angular, документация [Электронный ресурс]. – Режим доступа: https://angular.io/docs, свободный – (дата обращения: 12.07.2020)
4. Руководство по Angular 10 [Электронный ресурс]. – Режим доступа: https://metanit.com/web/angular2/, свободный – (дата обращения: 13.07.2020)
5. Angular Tutorial [Электронный ресурс]. – Режим доступа: https://www.webdraftt.com/tutorial, свободный – (дата обращения: 14.07.2020)
6. Визуализация данных при помощи Angular и D3 [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/414785/, свободный – (дата обращения: 21.07.2020)