

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ГОРОДА МОСКВЫ ДОПОЛНИТЕЛЬНОГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ ЦЕНТР
ПРОФЕССИОНАЛЬНЫХ КВАЛИФИКАЦИЙ И СОДЕЙСТВИЯ
ТРУДОУСТРОЙСТВУ «ПРОФЕССИОНАЛ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к итоговой аттестационной работе на тему
«Разработка web-ресурса с использованием
технологий HTML, CSS, JavaScript, Tailwind, PHP, mySql»
на примере web-ресурса: <https://sanchez.p-host.in/vs>

Публичный репозиторий GitHub проекта:

<https://github.com/Kristina212207/Diplom.git>

слушателя Сталидзанс Кристины Витальевны группы №: 0360
программы профессиональной переподготовки
«Frontend разработка»

Москва, 2023

Содержание

Содержание.....	2
Введение.....	3
Обзор функциональности проекта	3
Обзор технической части проекта.....	7
GUI-часть	7
RНР-часть	12
База данных mySql	14
СПИСОК ЛИТЕРАТУРЫ	15

Введение

Тема итоговой аттестационной работы «Разработка web-ресурса с использованием технологий HTML, CSS, JavaScript, Tailwind, PHP, mySql».

Процесс создания web-ресурса будет включать в себя выполнение следующих последовательных этапов:

1. Определение назначения web-ресурса, его функционала.
2. Написание исходного программного кода и запуск web-ресурса.
 - 2.1. Разработка и создание прототипа серверной части (прототип php-скрипта)
 - 2.2. Разработка структуры и создание базы данных mySql
 - 2.3. Разработка и создание прототипа web-части (GUI)
 - 2.4. Доработка исходного кода и проекта в целом
3. Тестирование работоспособности web-ресурса: выявление существенных недостатков работы web-ресурса с последующим их устранением.

Обзор функциональности проекта

Проект представляет собой сайт косметолога. Персональная страница специалиста в области индустрии красоты. Сайт представляет собой программу, построенную по принципу SPA (Single Page Application), что означает загрузку в браузер на старте одного html-файла и ряда программных модулей на языке javascript. Исполнение javascript-кода в интерпретаторе браузера позволяет web-средствами создавать скорее классическую компьютерную программу, чем просто web-сайт. Широкое повсеместное применение на различных устройствах стандартизированных браузеров и интерпретаторов javascript позволяет SPA-приложениям (в том числе и

сайтам) получать не только более широкую аудиторию пользователей, но и огромное количество поддержки разного рода разработчиков в этой области.

На рисунке 1 представлен внешний вид главной страницы вышеобозначенного web-сайта косметолога.

Из основных компонентов, помимо самого контента и его верстки, следует отметить меню, предназначенное для перехода от страницы и к странице (см. "1" на рисунке 1). Однако обращение к этим пунктам меню не приводит к перезагрузке html-страницы. Именно вследствие особенностей SPA-архитектуры происходит лишь обновление требуемых визуальных блоков.

Также следует отметить форму для входа в личный кабинет клиента косметолога. Эта функциональность необходима для идентификации пользователя и обеспечения дальнейшей его самостоятельной записи на прием.

Для целей тестирования предлагается создать пару пользователей



Рисунок 1

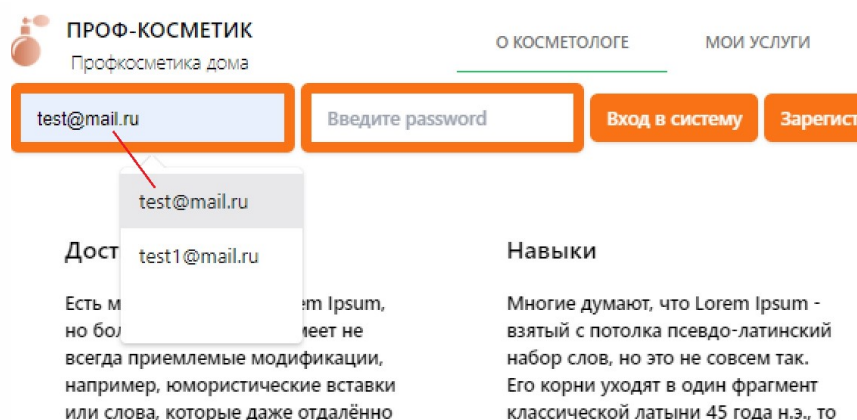


Рисунок 2

(зарегистрировать самостоятельно) или использовать уже имеющихся в базе данных сайта (см. рисунок 2; пароль для тестовых пользователей "1234"). Создание (самостоятельная регистрация) новых пользователей производится без ограничений.

После входа пользователя в систему для оценки функциональности проекта предлагается перейти в раздел (пункт меню) "Запись". В этом разделе пользователю предоставляется возможность осуществить самостоятельную запись на прием в один из 5 слотов (в течение каждого дня)

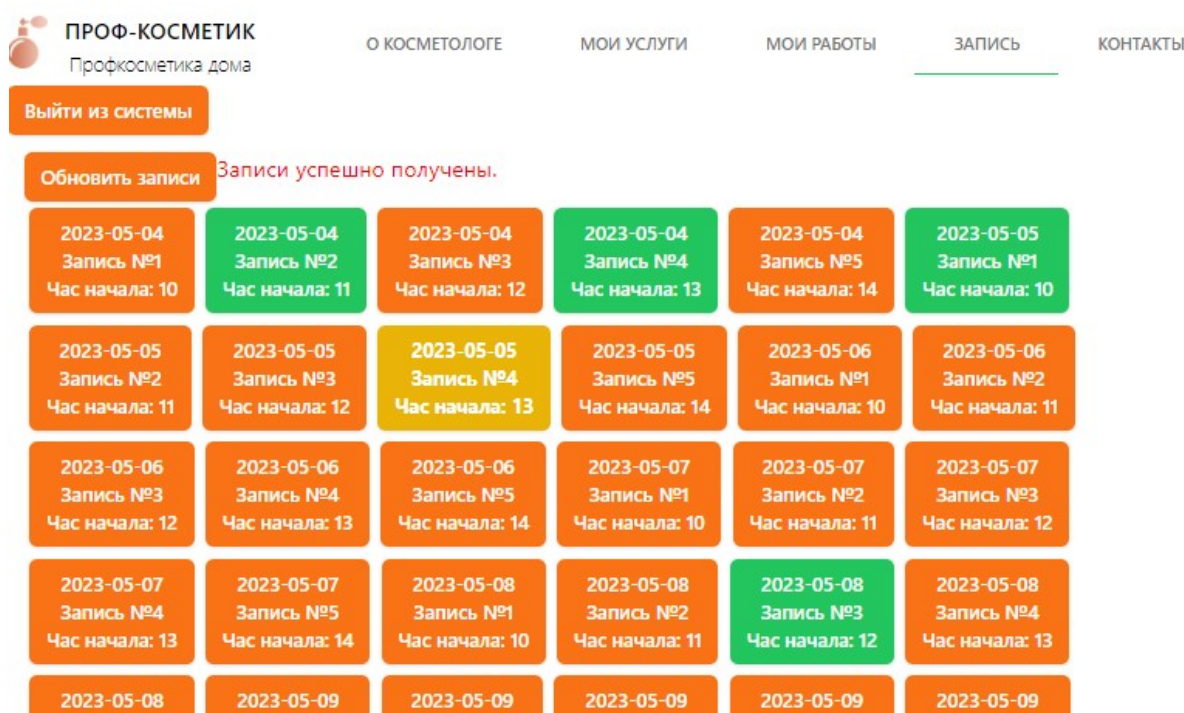


Рисунок 3

на следующие две недели от текущей даты (дата изменяется динамически, определяется на стороне сервера). Уже осуществленные записи данным пользователем отображаются зеленым цветом. Записи, осуществленные другими пользователями - желтым. Свободные слоты - оранжевым (см. рисунок 3).

Нажатие на свободном слоте приводит к попытке записи текущего пользователя на прием. В случае успеха или ошибки будет отображена соответствующая информация.

Нажатие на занятом слоте приводит к следующему: если слот занят текущим залогиненным пользователем, произойдет попытка удаления записи, то есть освобождения занятого текущим пользователем слота. Если слот занят другим пользователем, произойдет такая же попытка отменить запись другого пользователя, но из-за отказа сервером по причине отсутствия такого права чужая запись не будет отменена.

Для целей тестирования предлагается осуществить несколько записей

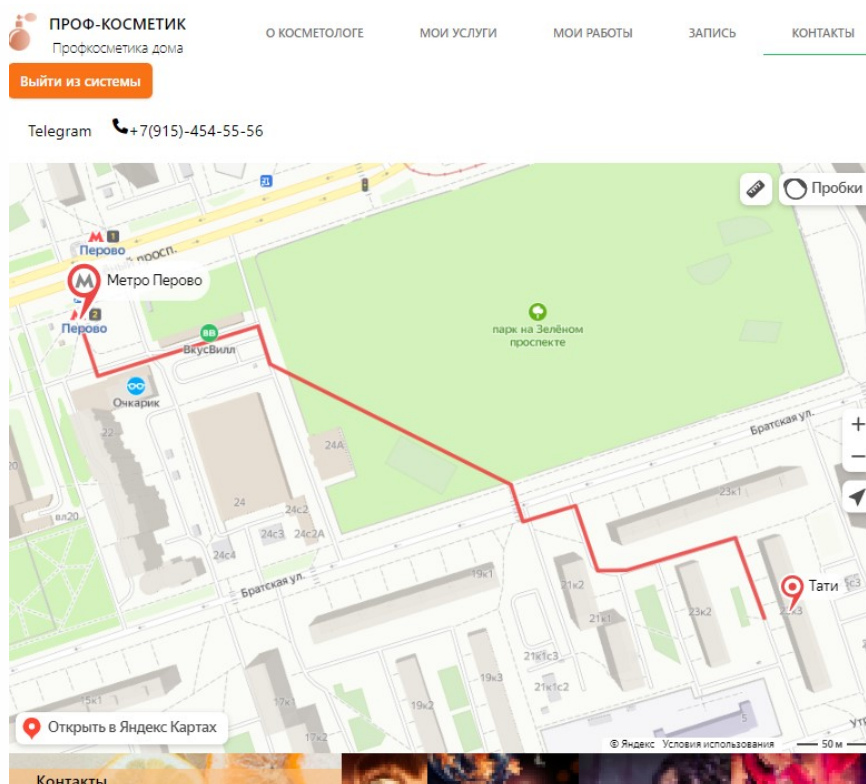


Рисунок 4

одним пользователем. Выйти из системы и зайти в нее заново под другим пользователем. Убедиться в наличии описанной функциональности системы.

На странице "Контакты" представлена интегрированная в GUI (web-сайт) карта с маршрутом от Яндексa. Так же имеются интерактивные ссылки на социальные сети (Телеграм) косметолога и номер телефона специалиста (см. рисунок 4).

Обзор технической части проекта

Проект состоит из трех частей: GUI-часть (она же web-сайт), PHP-часть (она же серверная часть) и база данных (БД mySql).

GUI-часть

Сайт построен на основе динамической генерации HTML javascript'ом. Код javascript разбит на блоки, которые подобны компонентам. Каждый компонент генерит HTML в свой тег. Этот тег - div, он имеет id="<имя_компонента/блока>Container".

Сайт построен на стилизованных div'ах с использованием фреймворка Tailwind. Этот фреймворк гармонично дополняет генерацию html в javascript, так как представляет собой модуль на javascript, который неким подобным же образом генерит css (динамически во время выполнения javascript'a браузером).


Блоки javascript делятся на два слоя. Одна часть блоков отвечает за работу с серверной частью сайта, а именно с php-скриптом main.php, расположенным и выполняющемся на сервере. Эта часть блоков - блоки или компоненты данных (компоненты data). К таким блокам или компонентам данных относятся следующие: loginData.js, recordsData.js (или просто loginData и recordsData).

Другая оставшаяся часть блоков/компонентов - блоки или компоненты GUI (Graphical User Interface), они же компоненты графического интерфейса пользователя. Графический интерфейс пользователя сайта построен на основе интерпретации HTML, CSS, Javascript браузером. К компонентам графического интерфейса относятся следующие: main.js, menu.js, login.js, records.js, about.js, gallery.js, services.js, contacts.js, body.js (или просто main, menu, login, records, about, gallery, services, contacts, body).

У всех компонентов (data и gui) имеется функция `<имя_компонента>_refresh()` без аргументов. Далее по тексту функция `<имя_компонента>_refresh()` будет называться просто `refresh`. Функция `refresh` запускает процесс обновления состояния компонента. Процесс начинается с http-запроса к серверу, точнее к расположенному на сервере скрипту `main.php`. Запрос к серверу инициируется стандартной функцией javascript браузера, функцией `fetch` (см. "1" на листинге 1).

Возврат из функции `refresh` компонентов `data` производится мгновенно,

Листинг 1

```
function recordsData_refresh() {  
  const url = "/vs/getRecords";  
  fetch(url)  1  2  
  .then(response => response.json())  3  
  .then(arr => { // arr - массив объектов записей  
  
    recordsData_obj = {  
      success: true,  
      records: arr,  
      message: "Записи успешно получены."  
    };  
  
  })  
  .catch(error => {  
    console.log(`getRecords ERROR: ${error.message}`);  
    recordsData_obj = {  
      success: false,  
      message: error.message  
    };  
  })  
  .then(() =>  4 eventEmitter.dispatchEvent(new Event("recordsData_refreshed")));  
}
```


сразу после инициации http-запроса к серверу. Такой мгновенный возврат из функции `refresh`, не дожидаясь завершения процесса получения данных с сервера и дальнейшего обновления состояния компонента, возможен за счет использования асинхронных возможностей `javascript`. Функция `fetch`, еще не получив результата, то есть ответа от сервера, сразу после своего вызова возвращает специальный объект, так называемый `Promise`, объект-обещание, позволяя к этому объекту-обещанию цеплять дальнейшие программные куски кода, которые будут выполнены после исполнения обещания (в данном случае обещание состоит в получении `http`-ответа от сервера в ответ на запрос). Если ответ от сервера не удалось получить, будет выполнен один кусок кода (функция, передаваемая в качестве аргумента методу `Promise.catch`). Если ответ от сервера был получен, то будет выполнен другой кусок кода (функция, передаваемая в качестве аргумента методу `Promise.then`). Функция, переданная в качестве аргумента методу `Promise.catch`, в случае своего дальнейшего исполнения, получит в качестве аргумента объект, содержащий описание произошедшей ошибки. А функция, переданная в качестве аргумента методу `Promise.then`, в случае если ошибки не произойдет, получит в качестве аргумента оговоренное объектом-обещанием значение. Метод `fetch` передает той функции, которая была передана в качестве аргумента методу `then` его `Promise`'а, `http`-ответ от сервера. В данном проекте подразумевается, что строка `http`-ответа от сервера представляет собой форматированную в соответствии с правилом стандарта `JSON` конструкцию (строку, вернее объект `Response`, см. "2" на рисунке-листинге выше, содержащий эту строку). Строка ответа затем разбирается (парсится) на части и собирается в `javascript`-объект, соответствующий по форме и содержанию `JSON`-строке из ответа сервера (см. "3").

По возврату `http`-ответа от сервера (от `main.php`) компонент `data` пытается преобразовать строку тела ответа в объект `javascript`. Ошибка обращения к серверу или преобразования строки ответа к объекту `javascript` отлавливается и обрабатывается компонентом `data`. По завершению анализа

возвращенного сервером объекта javascript или в случае ошибки после ее обработки компонент data создает (поднимает) событие (event) вида строки "<имя_компонента>_refreshed" (см. "4" на рисунке-листинге выше). Для диспетчеризации (подписки, поднятия и рассылки) событий в index.html создается глобальная переменная eventEmitter, содержащая экземпляр класса (функции-конструктора) EventTarget.

Глобальный экземпляр EventTarget'a обеспечивает связь одних компонентов с другими. Когда изменяется состояние какого-то одного компонента, другие заинтересованные в этом изменении компоненты узнают об этом изменении через свою подписку (прослушивание) на эти интересующие их события в этом глобальном eventEmitter. "Emitter" в названии переменной (emit - испускать, распространять) означает способность EventTarget'a распространять (публиковать) поднятые события своим подписчикам (слушателям, listener'ам).

Подписка на прослушивание события осуществляется вызовом метода EventTarget.addEventListener. Инициация распространения события среди подписчиков осуществляется методом EventTarget.dispatchEvent. При возникновении события "<имя_компонента>_refreshed" заинтересованные в новом состоянии, изменившегося компонента, блоки (другие компоненты) вызывают процедуру обновления своего состояния (свою функцию refresh, то есть обновления себя).

Данная модель компонентов представляет собой зачаточное состояние нормального компонентного фреймворка. Компоненты жестко связаны друг с другом, существуют в одном экземпляре, и как я узнала на собственном опыте, любые изменения в поведении сайта (компонентов) следует продумывать, отслеживать, проверять и перепроверять. Но тем не менее сама идея построения сайта из компонентов на javascript, которые генерят html небольшими, удобными для понимания и изменения кусками, мне очень понравилась, как идея, лежащая в основе таких фреймворков, как React, которого мы немного коснулись на нашем курсе.

Листинг 2

```
4  function login_refresh() {
5
6      if (typeof loginContainer === 'undefined')
7          return;
8
9      if (loginData_obj.success == false) {
10
11          loginContainer.innerHTML = `Не удалось получить информацию о сессии или запрос к серверу не удался`;
12
13      } else { // loginData_obj.success == true
14
15          if (loginData_obj.isLogged == true) { // залогинен
16              loginContainer.innerHTML = `
17                  <form onsubmit="return false;">
18                      ${buttonPiece_render("submit", "Выйти из системы", "login_logout()")}
19                  </form>
20                  <div id="login_loginInfo" style="color: red"></div>
21              `;
22          } else { // не залогинен, т.е. loginData_obj.isLogged == false
23
24              if (typeof login_loginInput !== 'undefined') {
25
26                  if (typeof loginData_obj.message !== 'undefined') {
27                      login_loginInfo.innerHTML = loginData_obj.message;
28                  }
29
30                  return;
31
32              };
33
34              loginContainer.innerHTML = `
35                  <form onsubmit="return false;">
36                      ${inputPiece_render("login_loginInput", "text", "Введите email")}
37                      ${inputPiece_render("login_passInput", "password", "Введите password")}
38                      ${buttonPiece_render("submit", "Вход в систему", "login_doLogin()")}
39                      ${buttonPiece_render("button", "Зарегистрироваться", "login_addUser()")}
40                  </form>
41                  <div id="login_loginInfo" style="color: red"></div>
42              `;
43
44          }
45      }
46  }
47
48  emitter.dispatchEvent(new Event("login_refreshed"));
49  }
```

Генерация HTML из полученных и преобразованных в объекты javascript ответов сервера, производится путем логической обработки значений полей этих объектов с дальнейшим переприсвоением значения свойству innerHTML элементу DOM контейнера компонента (см. например, листинг 2 строка 34).

Помимо попытки выделить из структуры кода ряд компонентов, для упрощения генерации HTML в javascript пришлось создать несколько вспомогательных функций, объединенных под общим названием или серией

- pieces (см. пример вызова этих вспомогательных функций, листинг 2, строки 36-39).

PHP-часть

Серверная часть представлена скриптом main.php. Настройками файла .htaccess веб-сервера Apache на хостинге, а именно правилом RewriteRule ^.*\$ main.php, все запросы к URL https://sanchez.p-host.in/vs/ и к аналогичному URL по протоколу http перенаправляются на обработку скрипту main.php. Так же к этому скрипту перенаправляются все запросы к этому серверу (к этому хосту), URL которых начинается с пути /vs и глубже.

Скрипт main.php для начинающихся с "/vs/site" URL-путей пытается вернуть в качестве ответа содержимое файла, лежащего в файловой системе сервера в одноименной с URL-путем папке site и глубже (см. "1" на листинге 3). Таким образом main.php осуществляет функцию по передаче требуемых статических (виртуально неизменяемых) файлов, таких как index.html и картинки.

Листинг 3

```
1  <?php
2
3  session_set_cookie_params(0, "/vs/");
4  session_start();
5
6  if (rtrim($_SERVER['REDIRECT_URL'], "/") == "/vs") { // если запрос хочет корень сайта
7
8      echo file_get_contents("site/index.html");
9
10 } else if (stripos($_SERVER['REDIRECT_URL'], "/vs/site") === 0) { // если запрос хочет какой-то файл из "папки" site
11
12     $filePath = substr($_SERVER['REDIRECT_URL'], 4, strlen($_SERVER['REDIRECT_URL']) - 4);
13
14     echo file_get_contents($filePath);
15
16 }
17
18 /* ----- */
19 /* /vs/addUser */
20 /* ----- */
21 else if (rtrim($_SERVER['REDIRECT_URL'], "/") == "/vs/addUser") {
22
23     if ($_SESSION["email"] != null) {
24
25         echo <<<JSON
26         {
27             "success": false,
28             "message": "Уже залогинен"
29         }
30     }
```

В случае, если URL запроса начинается не с /vs/site, скрипт main.php осуществляет функцию исполнения программного кода веб-сервиса (веб-службы). Конечно, если URL запроса совпадет с URL обслуживаемых скриптом main.php (см. "2" на листинге 3), таких как, например:

- <https://sanchez.p-host.in/vs/getLoginInfo> - endpoint для получения информации о состоянии залогиненности пользователя. Состояние залогиненности пользователя хранится в сессии PHP на сервере, а именно в файлах вида `sess_72258cb859bde6f4da9de9763d124bae`, расположенных в папке /tmp сервера.

Залогиненность пользователя обеспечивается сохранением в этом файле сессионной переменной email с непустым значением.

В случае отсутствия переменной email в PHP-сессии пользователя, осуществляющего http-запрос к серверу, или в случае пустого значения этой переменной или пустой строки, такой пользователь считается не залогиненным.

Сопоставление сервером http-запроса сессии (файлу в папке /tmp) производится по ключу, автоматически передаваемому браузером клиента - "куки" (cookie) PHPSESSID, значение которого должно совпадать с частью имени файла сессии (после префикса sess_).

- <https://sanchez.p-host.in/vs/logout> - endpoint для передачи серверу команды на разлогинивание текущего пользователя;
- <https://sanchez.p-host.in/vs/getRecords> - endpoint для получения списка записей (занятых слотов) от текущей даты на сервере на две недели вперед.
- и т.п.

Примечание. endpoint - метод веб-службы, иными словами функция. Не путать с методом протокола HTTP, который является параметром http-запроса и/или http-ответа. Для всех запросов проекта подразумевается использование метода http GET, но и другие методы не запрещаются, хотя и не проверялись.

База данных mySql

Для энергонезависимого эффективного хранения информации в проекте предусмотрено существование и использование реляционной базы данных (БД) на основе системы управления базами данных (СУБД) mySql. БД проекта состоит из двух взаимосвязанных таблиц vs_user и vs_record. Таблица vs_user предназначена для хранения списка пользователей проекта. Таблица vs_record предназначена для хранения списка записей (занятых слотов) пользователей проекта (web-сайта косметолога).

Таблица vs_record зависит от таблицы vs_user. Каждая запись (строка) в таблице vs_record должна ссылаться на существующую запись (строку) в таблице vs_user. Удалить запись из таблицы vs_user при существующих ссылающихся на нее записях таблицы vs_record невозможно (не даст сделать СУБД). Такое отношение таблиц в реляционной СУБД обеспечивается созданием в БД ограничения (constraint) вида Foreign Key (FK или "заимствованный ключ").

В проект включены скрипты для удаления и создания базы данных проекта с нуля, а также для заполнения базы данных некоторыми тестовыми данными (см. листинг 4). Для исполнения скриптов использовалась интегрированная среда разработки DBeaver.

Листинг 4

```
CREATE TABLE vs_user (  
    email VARCHAR(255),  
    pass VARCHAR(255),  
    PRIMARY KEY (email)  
);  
  
CREATE TABLE vs_record (  
    recordDate DATE,  
    startHour TINYINT,  
    email VARCHAR(255) NOT NULL,  
    PRIMARY KEY (recordDate, startHour),  
    FOREIGN KEY (email) REFERENCES vs_user (email)  
);
```

СПИСОК ЛИТЕРАТУРЫ

1. Сайт

Документация по фреймворку Tailwind [Электронный ресурс]: офиц. сайт. URL: <https://tailwindcss.com/> (Дата обращения: 03.05.2023).

2. Сайт

Официальная документация Яндекс по подключению API [Электронный ресурс]: офиц. сайт. URL: <https://yandex.ru/dev/maps/jsapi/doc/2.1/quick-start/index.html> (Дата обращения: 03.05.2023).

3. Сайт

Официальная документация ресурса Fontawesome [Электронный ресурс]: офиц. сайт. URL: <https://fontawesome.com/> (Дата обращения: 03.05.2023).

4. Сайт

Электронный учебник по HTML и CSS [Электронный ресурс]: офиц. сайт. <http://htmlbook.ru/html5> (Дата обращения: 03.05.2023).

5. Сайт

Официальная документация Mozilla [Электронный ресурс]: офиц. сайт. URL: <https://developer.mozilla.org/ru/> (Дата обращения: 03.05.2023).

6. Сайт

Официальная документация PHP [Электронный ресурс]: офиц. сайт. URL: <https://www.php.net/docs.php> (Дата обращения: 03.05.2023).

7. Сайт

Сборник курсов по IT [Электронный ресурс]: офиц. сайт. URL: <https://www.w3schools.com/> (Дата обращения: 03.05.2023).