**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ФИЗИКИ

КАФЕДРА РАДИОАСТРОНОМИИ

Направление: 03.03.03. — «Радиофизика»

КУРСОВАЯ РАБОТА

**ВЕБ-ПРИЛОЖЕНИЕ, ОТОБРАЖАЮЩЕЕ ТЕПЛОВУЮ КАРТУ ВРЕМЕНИ ВЗГЛЯДА ПОЛЬЗОВАТЕЛЯ НА ОБЛАСТИ ВЕБ-ДОКУМЕНТА, ПОСРЕДСТВОМ ФРЕЙМВОРКА ANGULARJS**

**Работа завершена:**

"\_\_\_"\_\_\_\_\_\_\_\_2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(A.A Ямолдин)

**Работа допущена к защите:**

Научный руководитель

ассистент

"\_\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(К.В. Скобельцин)

Заведующий кафедрой

канд. физ.-мат. наук, доцент

"\_\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(А.Д. Акчурин)

Казань — 2020

**СОДЕРЖАНИЕ**

Введение.....……………………………………………………………………….….3

ГЛАВА 1. СВОБОДНЫЙ БРАУЗЕРНЫЙ АЙТРЕКЕР WEBGAZER.JS...............4

1.1. Особенности айтрекера WebGazer.js……………….......……..........…..6

1.2. Достоинства и недостатки айтрекера WebGazer.js……........……...…10

ГЛАВА 2. СВОБОДНАЯ БИБЛИОТЕКА HEATMAP.JS......................................

2.1 Особенности библиотеки heatmap.js

2.2 Достоинтсва и недостатки heatmap.js

ГЛАВА 3. ОСНОВНЫЕ СВЕДЕНИЯ О СОЗДАНИИ ОДНОСТРАНИЧНЫХ ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ ФРЕЙМВОРКА ANGULARJS

3.1 AngularJS и совместимые языки программирования..........................

3.2 Среда разработки WebStorm IDE...............................................

3.3. Структура AngularJS приложений в среде WebStorm IDE для фреймворка AngularJS.......................................................15

ГЛАВА 4. ПРАКТИЧЕСКАЯ ЧАСТЬ…………………………..….……

4.1 Подготовка к работе приложения в index.html ………………21

4.2 Создание AngularJS приложения..............................

4.3 Запуск айтрекера WebGazer.js...................................

4.4 Калибровка WebGazer.js……………………………………..…………22

4.5 Создание тепловой карты с помощью heatmap.js.................................

4.6 Деплой приложения на хостинг GitHub Pages............................

Заключение…………………………………………………………….……..…..30

Список использованных источников………………………………….…….….31

Приложение………………………………………………………………………32

**ВВЕДЕНИЕ**

При разработке современных веб-приложений часто требуется сбор данных о действиях пользователя в режиме реального времени, особенно интересны места, на которых наиболее часто падает взгляд пользователя, так как 80-85% информации воспринимается через визуальный контакт, а 37% маркетологов назвали визуальный контент самым главным иснтрументом контент-маркетинга для бизнеса.[1].

Мы решили визуализировать время нахождения взгяда на какой-ибо области веб-страницы, создав приложение, которое используя вебкамеру в режиме реального времени строит тепловую карту времени нахождения взгяда в какой-либо области веб-документа. Данное приложение может быть использовано так стремительно развивающимися в нынешнее время интернет магазинами, данный факт отбрасывает все вопросы об актуальности данной работы. К примеру, зная, на каком месте дисплея взгяд пользователя задерживается дольше обычного, интернет ретейлеры смогут помещать товар, в продаже которого особо заинтересованы, в наиболее просматриваемое место интернет страницы, что приведет к увеличению продаж данного продукта. В настоящий момент для сбора данных существует несколько готовых решений, такие как JavaScript-библиотека TensorFlow[2], JavaScript-библиотека WebGazer.js[3]. При работе таких библиотек часто требуется передача цифровых данных, полученных от взаимодействия пользователя с веб-страницей, посредством различных датчиков, либо приборов на персональный компьютер.

Поэтому перед нами была поставлена цель осуществить передачу данных на компьютер с помощью созданного нами AngularJS веб-приложения, вебкамеры, дальнейшей их обработки с помощью свободной библиотеки WebGazer.js и наглядное представление этих данных в виде тепловой карты построенной посредством JavaScript-библиотеки heatmap.js.

Для достижения поставленной цели потребовалось решить следующие задачи:

* ознакомление со свободной библиотекой WebGazer.js;
* ознакомление со свободным скриптом heatmap.js
* изучение языка программирования JavaScript;
* изучения языка разметки HTML;
* Изучение языка таблиц стилей CSS
* Изучение фреймфорка AngularJS
* Работа с git и размещение приложения на хостинге GitHub Pages
* написание программы для внедрения библиотеки WebGazer.js в AngularJS web-приложение, а также для передачи данных в библиотеку отображения heatmap.js, нормальзации входных даннх на максимальное значение и посроение тепловой карты движения глаз по веб-странице

**ГЛАВА 1. СВОБОДНЫЙ БРАУЗЕРНЫЙ АЙТРЕКЕР WEBGAZER.JS**

**1.1.Особенности айтрекера WebGazer.js**

WebGazer.js - это свободно-предоставляемая JavaScript-библиотека для отслеживания местоположения взгяда пользователя на web-странице в режиме реального времени, использующая данные, полученные с помощью обычных веб-камер.

Webgazer.js разработана на основе исследования, проведеного Университетом Брауна[4]. Работа над файлом калибровки была разработана в контекте курсового проекта с целью улучшения обратной связи WebGazer.js, предложенного доктором Джеральдом Веббером и его командой: доктором Клеменсом Зейдлером и Кай-Чунг Леуном.

Модель отслеживания взгяла, используемая в этой библиотеке, самокалибруется путём наблюдения за взаимодействиями веб-пользователя с веб-страницей и оттачивает зависимости между особенностями положения глаз пользователя и его местоположением на мониторе.

WebGazer.js - библиотека, написанная полностью на JavaScript и с помощью всего нескольких строк кода может быть интегрирована в любой веб-сайт. Webgazer работает полностью на клиентской стороне браузера, что гарантирует безопасность и сохранность персональных данных, так как никая информация не сможет быть отправлена на сервер без согласия пользователя на предоставление прав доступа к пользовательской веб-камере.

**1.2.** **Достоинства и недостатки айтрекера WebGazer.js**

В данной работе была выбрана именно эта библиотека в силу общедоступности программного обеспечения и средств разработки. Кроме того, WebGazer.js делает простой работу с пользоваельскими данными для разработчика, и в отличие от иных систем, даёт ряд превосходств для преподавателей, студентов и радиолюбителей:

* **Легкость в использовании.** Самое большое преимущество

библиотеки Webgazer.js - это готовая к использованию структура. Всё, что требуется для работы библиотеки это влючение нескольких строк кода в шапку HTML страницы.

* **Широкая браузерная пожержка**.Еще одно большое преимущество

WebGazer.js-это возможность работы в большинстве современных браузеров за счет использования getUserMedia/Stream API[5] для доступа к веб-камере. Этот метод поддерживается большинством современных браузеров[6]

* **Легкий доступ к выходным данным библиотеки.** Для получения

предсказанных данных местоположения пользовательского взгляда достаточно добавить несколько строк JavaScript кода в тело HTML страницы.

* **Программное обеспечение с открытым исходным кодом.**  WebGazer.js

это библиотека с открытым исходным кодом, что дает большое преимущество для разработчка в плане понимания механизмов предсказания местоположения взгляда пользователя, реализованных в программе посредством просмотра “внутренностей” библиотеки.

* **Нет необходимости в специальном программном обеспечении**.

Большинство веб-камер поддерживаются данной библиотекой.

Но у WebGazer также есть и недостатки:

* **Структура.** При создании проекта требуется совершать межскриптовые

и межфункциональные переходы, а громоздкость библиотеки не позволяет осуществлять это быстро, а главное, зачастую теряется логическая связь между функциями, переменными и областями видимости.

* **Малочисленное сообщество**.В связи с малой извесностью данной

библиотеки в сети практически нет примеров использования WebGazer.js в веб-разработке, в связи с этим во многих аспектах программы приходилось быть пионером.

* **Скудность документации.** Официальная документация описывает лишь

малочисленные аспекты работы данной библиотеки.

* **Большой вес.** Объем данной библиотеки составляет 2.4МБ, что

значительно тормозит загрузку страницы на стороне пользователя.

* **Возвращенные координаты относительно окна просмотра.**

Предсказанные значения возвращаются относительно окна просмотра и не учитывают вертикальную и горизонтальную прокрутку страницы, что заставляет нас вносить собственную корректировку на прокрутку.

**ГЛАВА 2. СВОБОДНАЯ БИБЛИОТЕКА HEATMAP.JS**

**2.1. Особенности библиотеки heatmap.js**

heatmap.js - это свободно предоставляемая JavaScript- библиотека для визуализации трехмерных данных на веб-странице в режиме реального времени.

heatmap.js получает на вход трехмерные данные, две размерности- координаты точки (x, y) и третья размерность вес данной точки. В нашем случае все точки равноправны, поэтому имеют одинаковый вес.

Принцип построения тепловой карты состоит в следующем: если точка с координатами (x,y) и весом (value) в пределах какой-то области (radius) попадает в область уже существующей точки, то вес данной области будет равен сумме весов этих точек. Суммарное значение делится на максимальное, в резуальтате того, какую часть данная сумма занимает от максимального значения, выбирается каким цветом отображать данную область.

heatmap.js - это библиотека, написанная полностью на JavaScript и с помощью всего нескольких строк кода может быть интегрирована в любой веб-сайт. heatmap.js работает полностью на клиентской стороне браузера, что гарантирует безопасность и сохранность персональных данных.

**2.2. Достоинства и недостатки heatmap.js**

В данной работе была выбрана именно эта библиотека в силу общедоступности программного обеспечения, а также имеет ряд достоинств для преподавателей, студентов и радиолюбителей:

* **Бытрота.** Во время последнего обновления библиотеки был

добавлен новый модуль обновления данных, в результате чего имеется возможность построения тепловой карты в режиме реального времени.

* **Мощность.** Бибиотека позволяет обрабатывать до 40 000 точек.
* **Легкость в использовании.** Для получения возможнсти

использовать данную библиотеку достаточно включить несколько строк кода в шапку HTML страницы.

* **Легкий вес.** Объем данной библиотеки всего 25кБ, в результате чего

браузер тратит минимальное время на загрузку страницы.

Но у heatmap.js также есть и недостаток:

* **Отсутсвие нормировки на динамический максимум.** Одним из

самых больших недостатков данной библиотеки является нереализованная функция нормировки данных на максимальное значени, в результате этого нам пришлось реализовать этот функционал собственноручно.

**ГЛАВА 3. ОСНОВНЫЕ СВЕДЕНИЯ О СОЗДАНИИ ОДНОСТРАНИЧНЫХ ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ ФРЕЙМВОРКА ANGULARJS**

**3.1. AngularJS и совместимые языки программирования**

Для создания веб-приложений в настоящее время существует множество фреймворков: vue.js, react.js, AngularJS и т.д. Все вышеперечисленные фреймворки имеют схожие функции. Они были спроектированы для того, чтобы избавить разработчика от большого колличесва рутинной работы и существенно облегчить разработку[7]. В данной работе был выбран фреймворк AngularJS в силу использования им MVC (Model-View-Controller) шаблона, а также существования двухстороннего связывания, позволяющего динамически изменять и передавать данные из одного места веб-приложения в другое, синхронизируя таким образом модель и представление.

AngularJS является JavaScript фреймворком, главным языком программирования является JavaScript для динамической части приложения и HTML/CSS для разметки веб-страницы. JavaScript - это язык, программы которого выполняемы в разных средах. В данной работе речь будет идти о браузере и серверной платформе Node.js. Кроме языка JavaScript также используется “язык гипертекстовой разметки” (HTML - HyperText Makup Language). Этот язык программирования является стандартизованным языком разметки веб-страниц. Он интерптретируется браузерами и, полученный в результате текст, отображается на дисплее. Еще одиним используемым языком программирования в данной работе является СSS (Cascading Style Sheets - каскадные таблицы стилей). Это язык, описывающий внешний вид документа, написанного с помощью стандартизованного языка разметки веб-страниц (HTML).

**2.2 Среда разработки WebStorm IDE**

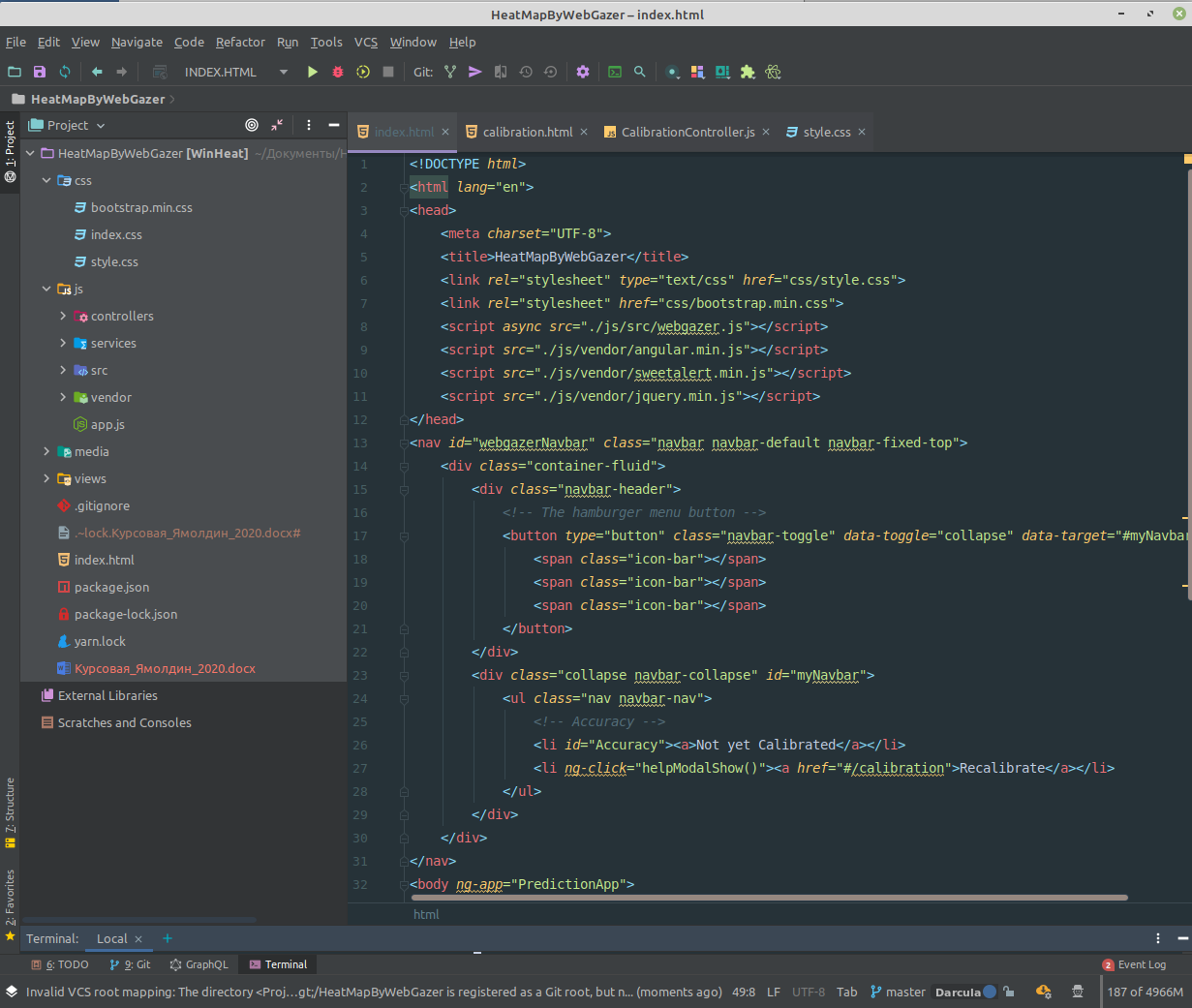


Рис. 1. Стартовое окно WebStorm IDE.

Интегрированная среда разработки WebStorm (IDE) - это кроссплатформенное приложение (для Windows, macOS, Linux), написанное на языке программирования Java. Исходный код для IDE выпущен под проприетарной лицензией от компании JetBrains. Важно отметить, что для студентов и преподавателей лицензия предоставляется абсолютно бесплатно. WebStorm IDE поддерживает языки JavaScript, CSS и HTML, используя специальные правила структурирования кода. Webstorm IDE имеет LiveEdit - возможность отслеживать “на лету” изменения на странице браузера при редактировании HTML, JavaScript или CSS кода проекта.

**3.3. Структура программ в среде WebStorm IDE для фреймворка AngularJS**

Программы в среде программирования WebStorm IDE называются проектами. Проекты, сделанные на AngularJS называются одностраничными приложениями (SPA - Single Page Application). Приложения AngularJS используют паттерн модель-представление-контроллер (MVC - Model-View-Controller). Компонент Model - обычно обозначает сущность, т.е. объект с набором характеристик, зачастую это файлы с данными на языке программирования JavaScript. Компонент View - файлы представления. Часто нам нужно предоставить информацию, содержащуюся в компоненте Model на обозрение пользователя, для этого во фреймворке AngularJS используется компонент View, который отображает файлы, написанные на языке программирования HTML, на дисплей. Компонент Controller - управляет потоком данных в приложении. Контроллеры взаимодействуют с данными модели и могут в зависимости от поставленных задач изменять их.

AngularJS приложения имеют общую структуру расположения файлов и папок, следование которой является примером хорошего тона в среде программистов- разработчиков.

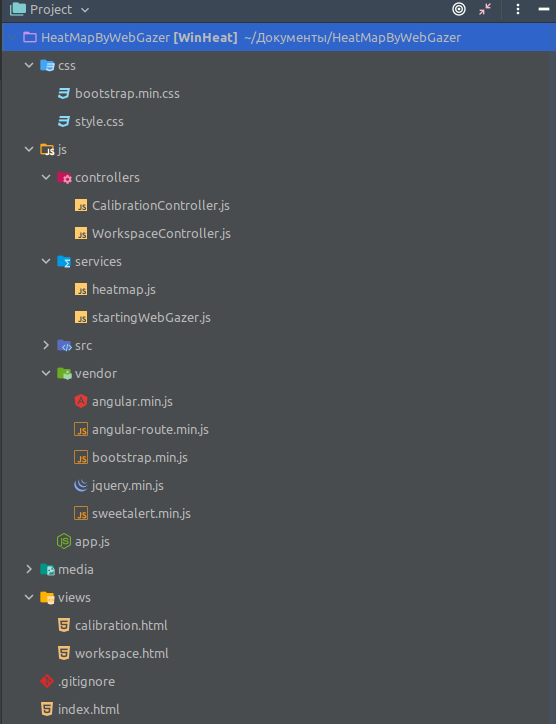


Рис.2. Общая структура приложения.

Файлы AngularJS приложения располагаются в некоторой общей директории, которая называется корневой. На рис.2 она выделена синим цветом и называется “HeatMapByWebGazer”. Внутри корневой директории располагаются поддиректории: css - директория, содержащая файлы, написанные на языке таблиц стилей и предназначенные для позиционирования и придания стилей для любого элемента из файлов компоненты View, написанных на HTML, js - директория, содержащая все файлы для нашего приложения, написанные на язые JavaScript, является корневой для app.js файла, внутри которого мы определяем AngularJS приложение, а также для поддиректорий controllers - включающей контроллеры, используемые в нашем приложении, services - включающей особые объекты или функции, выполняющие некоторые общие для всего приложения задачи, src - директория, включающая в себе любые дополнительные JavsScript файлы, которые мы хотели бы использовать в приложении, vendor (поставщик) - директория, содержащая непосредственно тело AngularJS приложения[8], а также некоторые дополнительные модули и системные файлы, используемые в приложении. Директория media - содержит мультимедиа файлы для нашего приложения, такие как видео, изображения, музыкальные файлы и т.п. Views - директория, содержащая HTML файлы представления приложения. Файл .gitignore - гит файл, опреляющий файлы исключения для гит. index.html - основной файл представления, который первый исполняется при загрузке страницы, внутри него определено AngularJS приложение, подключены внешние срипты, а также внутри него и происходит основное действие, по-сути, мы всегда находимся внутри этого файла, меняя только его внутренности, посредством взаимодействия с данными и представлениями с помощью AngularJS фреймворка. Именно поэтому приложения, написанные с помощью AngularJS являются SPA.

**ГЛАВА 4. ПРАКТИЧЕСКАЯ ЧАСТЬ**

**4.1 Подготовка к работе приложения в index.html**

Как было сказано выше, файл index.html выполняется перым при запуске AngularJS приложения, поэтому важно внутри него подключить все внешние сторонние JavaScript срипты, тело AngularJS приложения и файлы стилей (рис. 2.) , которые мы собираемся использовать в нашем приложении, а также используемые внутри него сервисы и контроллеры (рис. 3.).

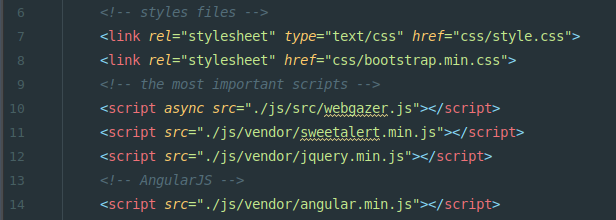


Рис. 2. Сторонние скрипты, файлы стилей и тело AngularJS.

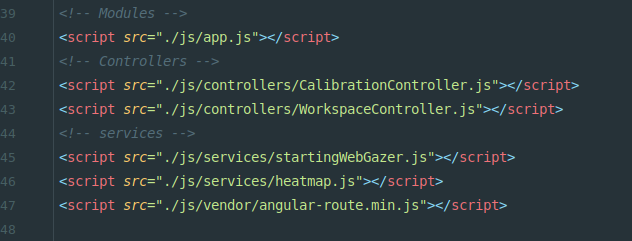


Рис. 3. Определение приложения, его сервисов и контроллеров.

Помимо этого в index.html нужно указать в какой зоне видимости документа будет выполняться наше приложение и в каких местах будут отображаться директивы. В нашем случае весь документ будет AngularJS приложением, поэтому мы его подключаем в теле докумета (рис. 4.), а также указываем место в котором будет отображаться представления для соответсвующих маршрутов (рис. 4.).



Рис.4. Подключение приложения и сооответвующего представления

**4.2 Создание AngularJS приложения**

Файл app.js является обязательным файлом AngularJS приложения.



Рис.5 . Вид файла app.js

Этот файл “говорит” браузеру, что это приложение AngularJS, имеющее названиe PredictionApp. В квадратных скобках указываются зависимости для данного приложения, иными словами, указываются некоторые дополнительные модули (функции), которых изначально нет в теле AngularJS приложения. Ниже пишется конфигурация приложения. В зависимости мы указали модуль ‘ngRoute’ который находится в файле angular-route.min.js[9] и позволяет использовать маршрутизацию внутри нашего приложения. Для конфигурации маршрутов используется объект $routeProvider, определенный внутри модуля ngRoute. Метод $routeProvider.when принимает два параметра: название маршрута и объект маршрута. Название маршрута это URL, по которому осуществляется доступ к представлению, находящемуся в templateUrl: и обрабатывающего его котроллеру в controller:. Метод $routeProvider.otherwise вызывается в том случае, если приложению не удалось найти сопоставить маршрут из $routeProvider.when и браузерным запросом и перенаправляет пользователя по маршруту, прописанному внутри redirectTo:.

**4.3 Запуск айтрекера WebGazer.js**

Для того чтобы начать использовать библиотеку WebGazer.js нужно сначала её запустить. Нами было принято решение вынести запускающий срипт в services (рис. 2), так как это действие является общим для всего приложения. Мы использовали реализацию с помощью сущности “factory” которая включает в себя функцию запуска трекера. Сервис представляет собой файл, написанный на языке программирования JavaScript. (рис. 6).

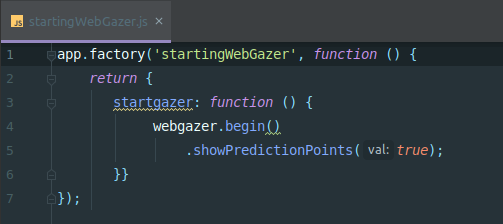


Рис. 6.

**4.3 Калибровка WebGazer.js**

Маршрут по умолчанию в соответвует названию ‘/calibration’ (рис.5), именно на него мы будем попадать при инициализации приложения. Этому маршруту сопоставлена калибровка айтрекера webgazer.js.

Как видно из рис.5 маршруту “/calibration” соотвествует собственный контроллер “CalibrationController” и собственное представление “templateUrl”. “CalibrationController” находится по пути “./js/controllers/CalibrationController.js” (рис. 2) и представляет собой файл, написанный на языке программирования JavaScript. Рабочие функции были предоставлены библиотекой WebGazer.js. Наша задача состояла в том, чтобы обеспечить работоспособность данных функций внутри соответсвующего контроллера. Для этого мы создали контроллер(рис. 7) внутри которого поместили рабочие функции, а также указали в зависимостях контроллера сервис: запускающий скрипт “startingWebGazer”. Это означает, что при вызове данного контроллера библиотека WebGazer.js будет подключаться автоматически.

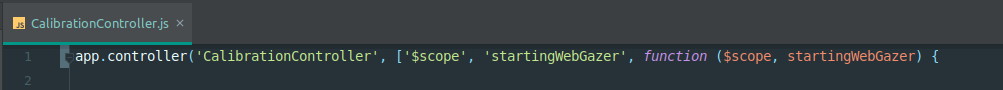


Рис. 7. Определение контролера.

После инициализации котроллера в указанное место для соответсвующего представления (рис. 4) вставится кусок HTML кода из соответсвующего templateUrl в результате чего на экране появится предложение откалибровать айтрекер с краткой инструкцией (рис.8).

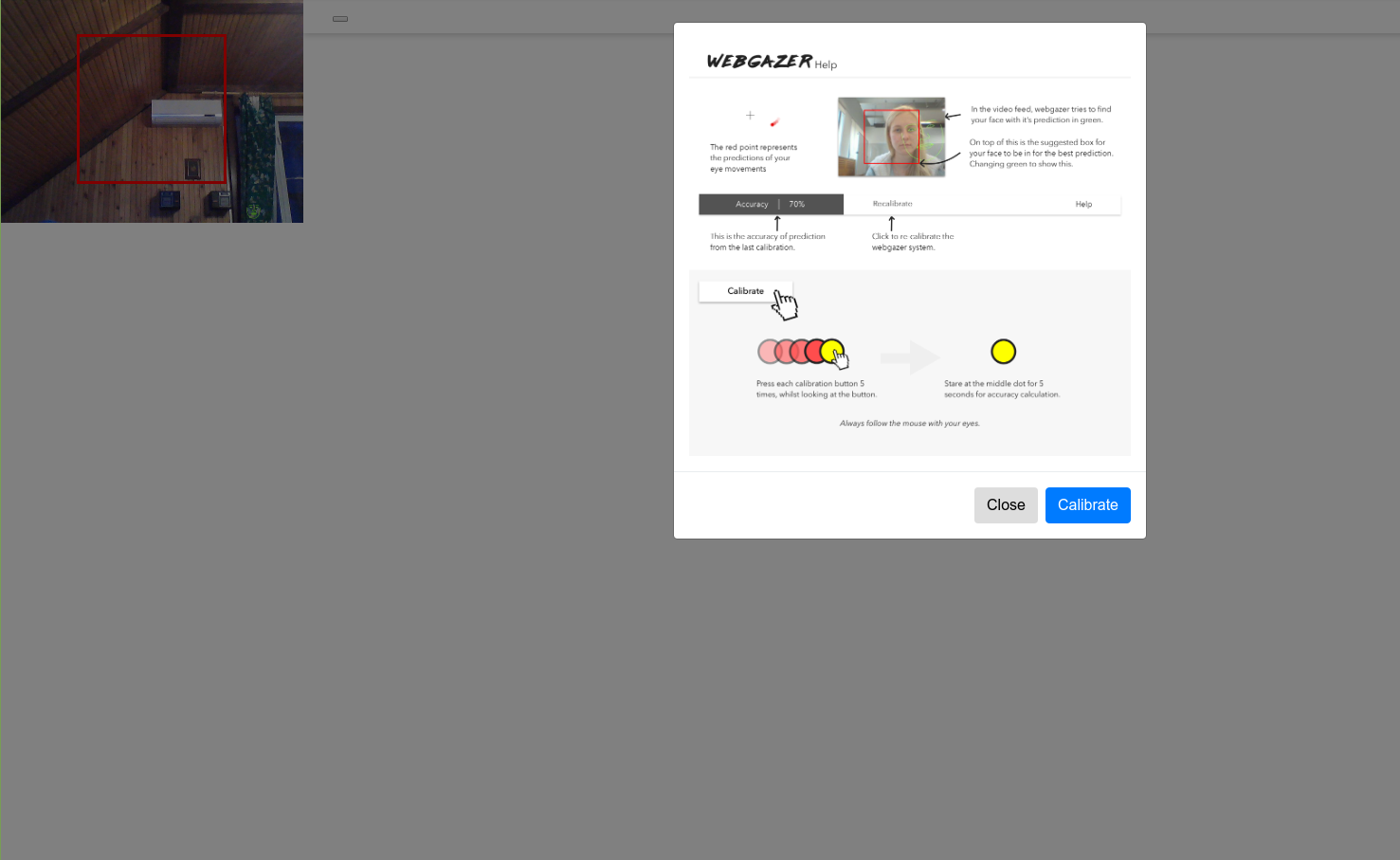


Рис.8. Калибровочная страница.

Для продолжения необходимо кликнуть на кнопку ‘Calibrate’. Далее для пользователя будет представлена ещё одна более подробная инструкция и порядком действий (рис. 9).

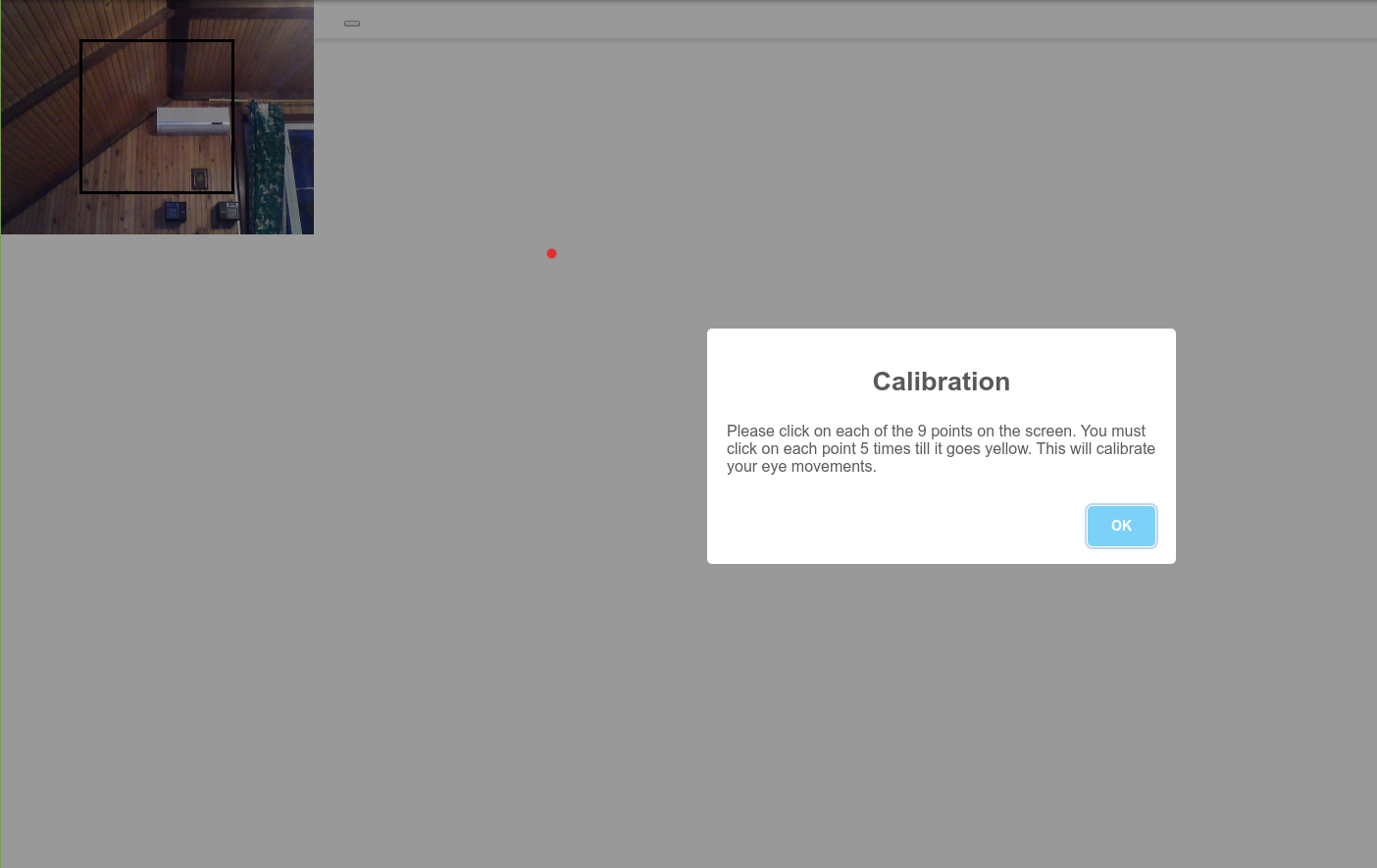


Рис.9. Подбробная инструкция по калибровке трекера.

Для калибровки трекера необходимо кликнуть 5 раз на каждую из предложенных окружностей (рис. 10).

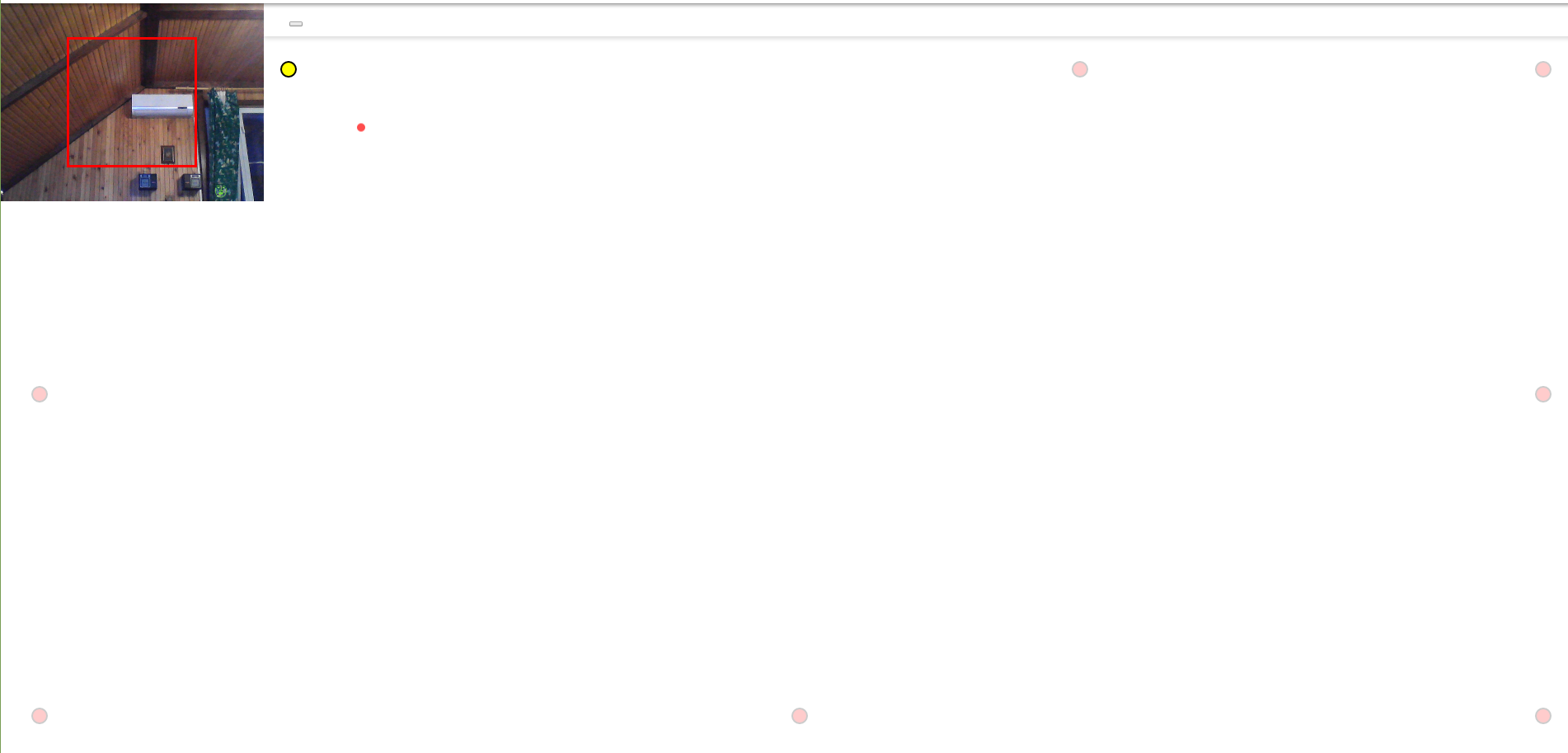


Рис.10. Прокликивание окружностей.

После прокликивания всех окружностей библиотека WebGazer.js с помощью внутренних функий вычислит меру точности калибровки и отобразит её на экран в виде вспылвающего окна (рис. 11).



Рис. 11. Мера точности калибровки.

На этом калибровка айтрекера WebGazer.js закончена. Пользователю предложено либо перекаблироваться, кливнув на кнопку “Recalibrate”, либо продолжить, кливнув на кнопку “Ok”. При клике на кнопку “Ok” функция window.location.href из CalibrationController.js (рис. 12) перенаправляет нас на маршрут ‘/workspace’, который в свою очередь имеет свой собственный контроллер WorkspaceController и соответсвующее ему представление в файле workspace.html.

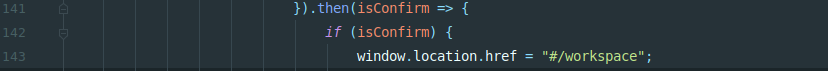


Рис.12. Функция перенаправления на маршрут ‘/workspace’.

**4.4 Создание тепловой карты с помощью heatmap.js**

Как видно из рис.5 маршруту “/workspace” соответствует контроллер “WorkspaceController” и представление по соответсвующему адресу из templateUrl. “WorkspaceController” находится по пути “./js/controllers/WorkspaceController.js” (рис. 2) и представляет собой файл, написанный на языке программирования JavaScript. Рабочие функции были предоставлены библиотекой heatmap.js. Наша задача состояла в том, чтобы обеспечить работоспособность данных функций внутри соответсвующего контроллера, наполнить его трехмерными данными и реализовать нормирование на максимальное значение. Для этого мы создали контроллер WorkspaceController (рис. 13), внутри которого проиниализировали массив datapoints, который будет содержать объекты, содержащие внутри себя координаты точки относительно левого верхнего края документа и её “вес”. В нашем случае все точки равнозначны, поэтому имеют одинаковый вес.

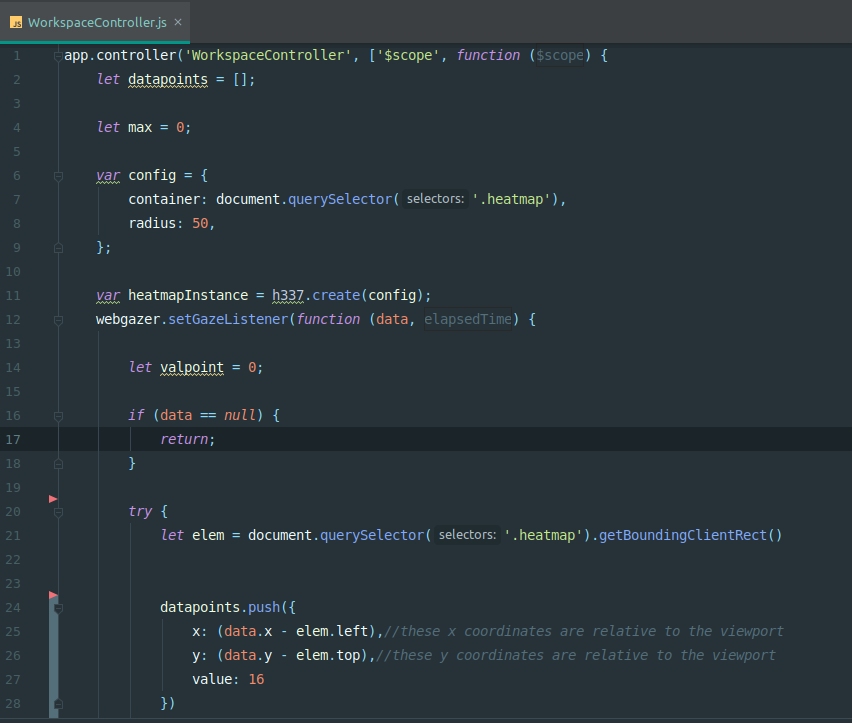


Рис. 13. Заполнение массива datapoints данными

Для того, чтобы выполнить нормировку на максимальное значение было решено при каждом добавлении новой точки проходить по всем предыдущим, уже содержащимся в массиве точек с помощью функции forEach() языка программирования JavaScript , в случае, если данная точка попадает в “квадрат” предыдущей, то “вес” данной области увеличивается на единицу. Увеличение области в 1/16 веса точки обусловлено тем, что в квадрат стороной, равной радиусу окружности можно упаковать максимум 4 окружности[10] этого радиуса. Как известно из теории вероятностей и комбинаторики, вероятность появления назависимых событий равно произведению их вероятностей, вероятность попадания окружности в квадрат равна 1\4, применяя это для четырех квадратов и, используя формулу вероятности независимых событий получаем вероятность, равную 1/16, что и отражено в увеличении веса участка.

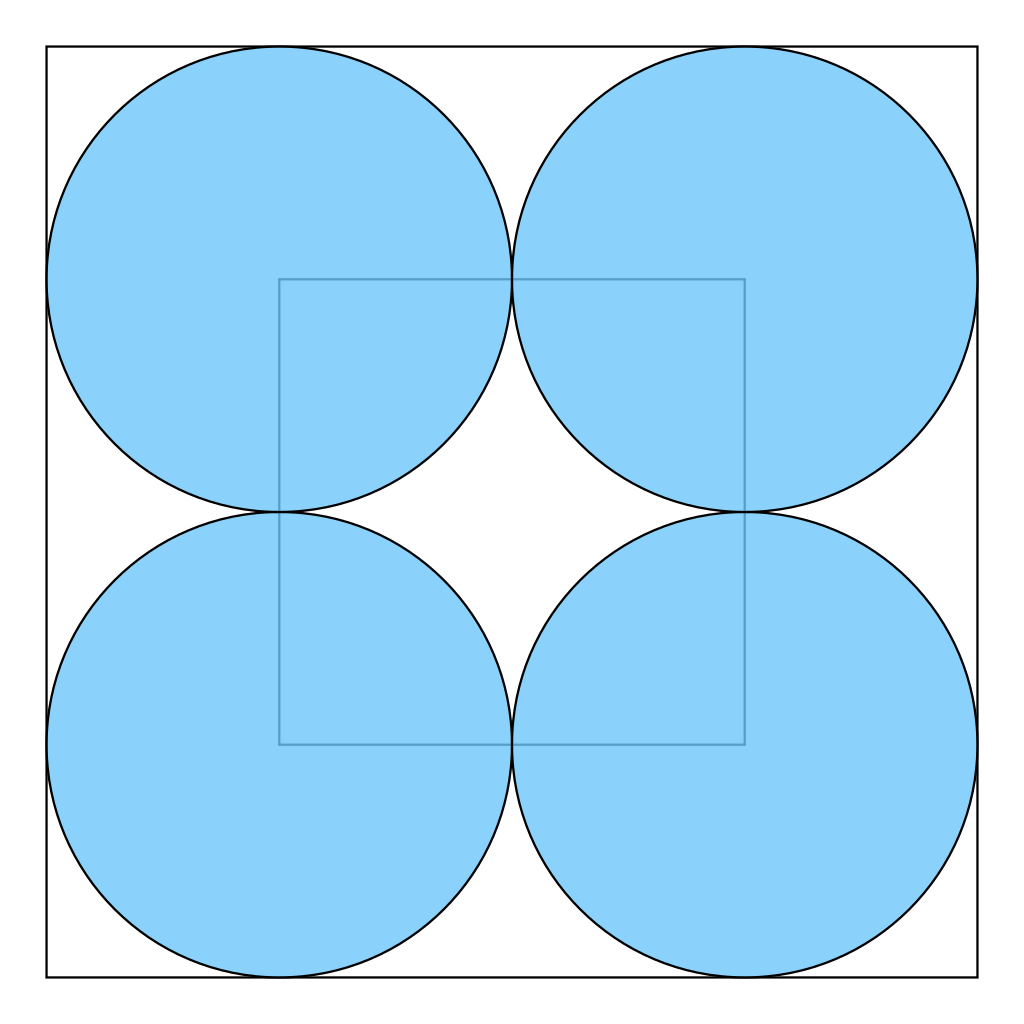


Рис. 14. Упаковка окружностей в квадрат.

С помощью метода Math.max(val1, val2) языка программирования JavaScript выбираем максимальное из двух значений и передаем в обновленный массив точек с обновленным максимумом на вход библиотечкой функции setData(data) библиотеки heatmap.js для обновления тепловой карты на экране пользователя(рис. 15).



Рис.15. Реализация пересчета максимума и передача данных на вход функции setData(data).

В результате работы нашего AngularJS приложения в режиме реального времени у нас есть возможность пронаблюдать как области на экране, на которых был взгляд пользователя, так и сделать вывод о том, на каких областях наиболее долго задерживал свой взляд пользователь.

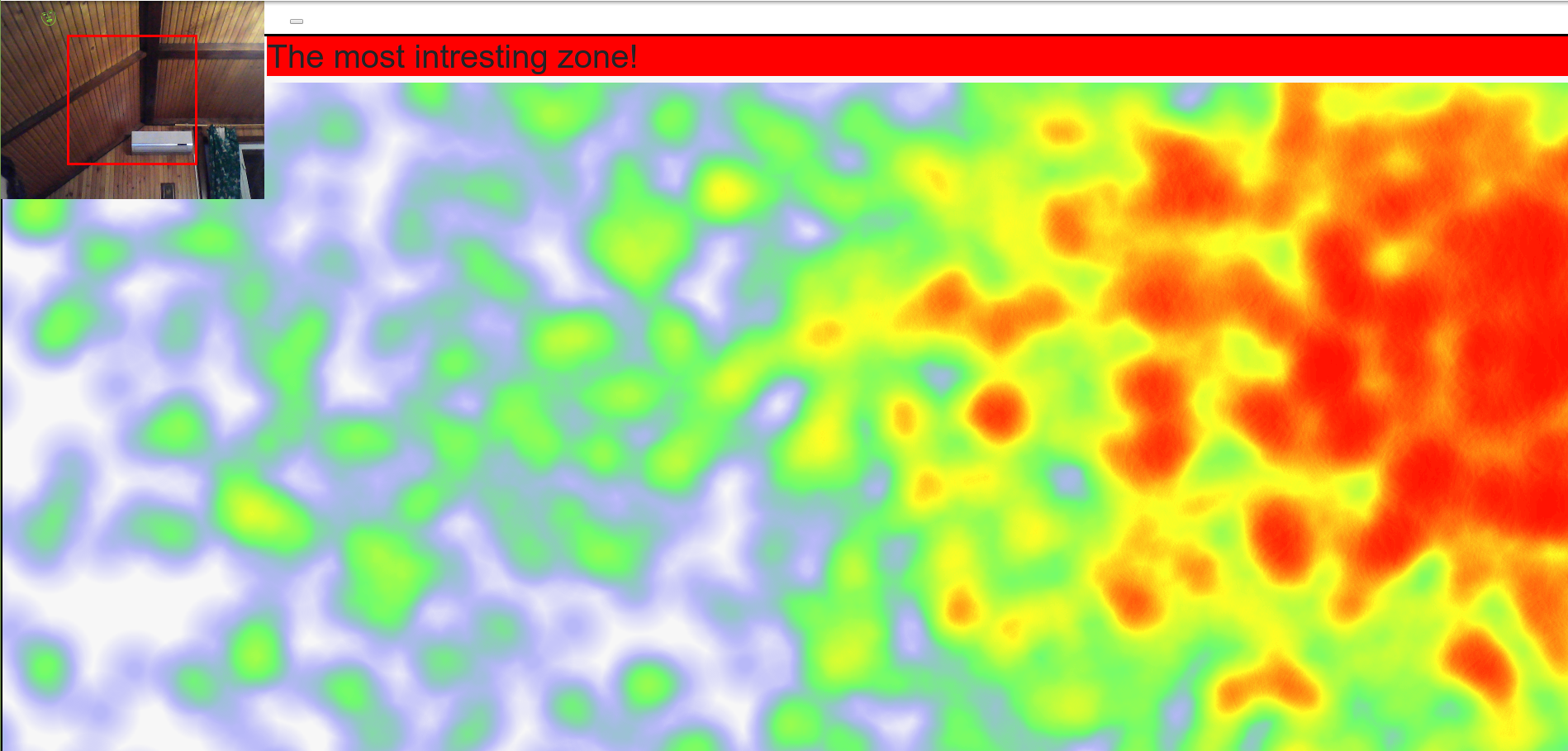


Рис. 16. Итоговый вид тепловой карты.

**4.6 Деплой приложения на хостинг GitHub Pages**

Для того чтобы разместить веб-прилодение в сети Интернет необходимо поместить его на веб-сервер. Хостинг GitHub Pages это бесплатная услуга по предоставлению ресурсов для размещения информации на сервере, постоянно имеющем доступ к сети Интернет. Для того чтобы воспользоваться этой услугой необходимо пройти регистрацию на сайте https://github.com/ (Рис. 17)

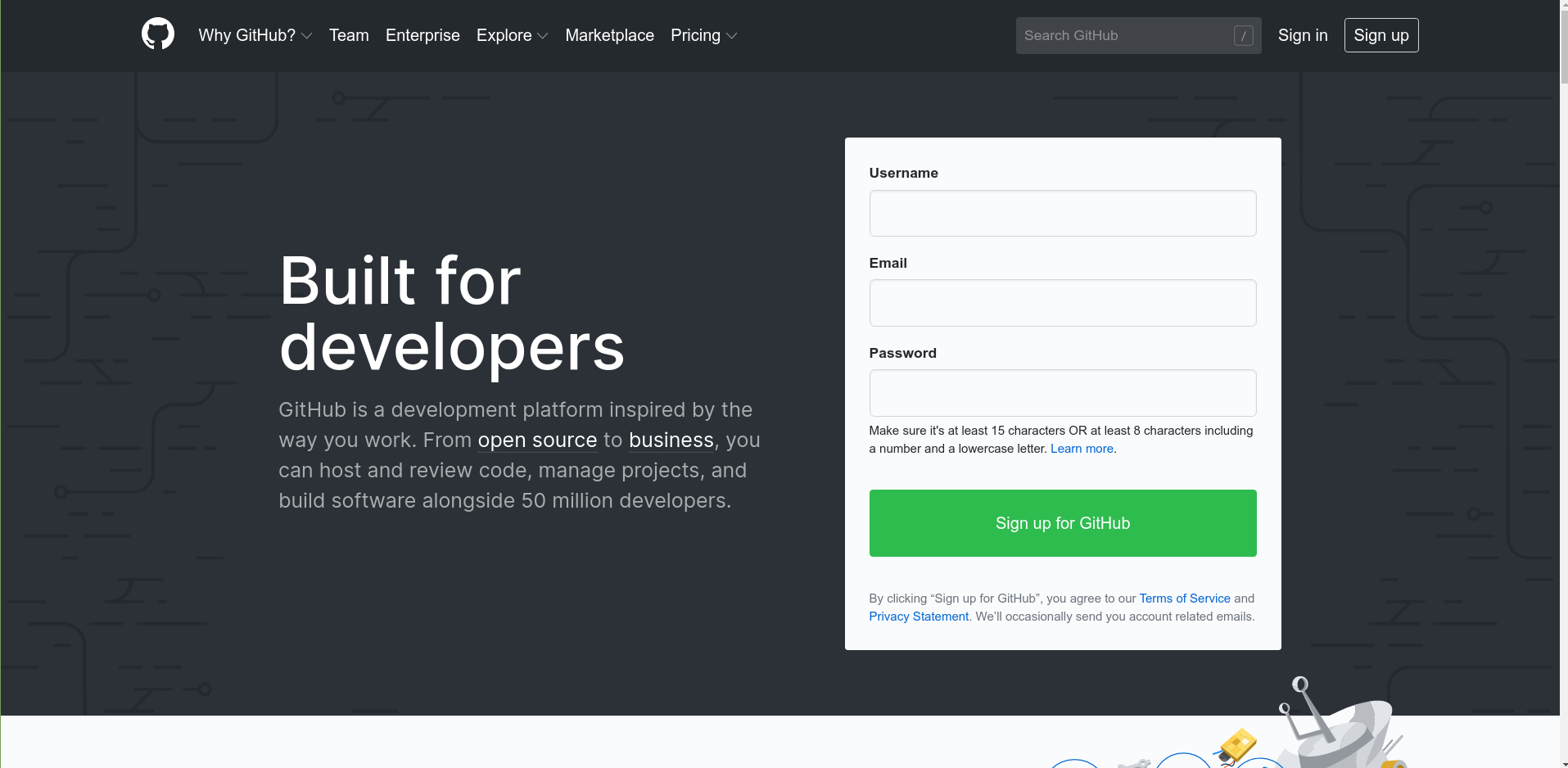


Рис.17 Регистрация на https://github.com/

После регистрации необходимо создать новый репозиторий (Рис.18)

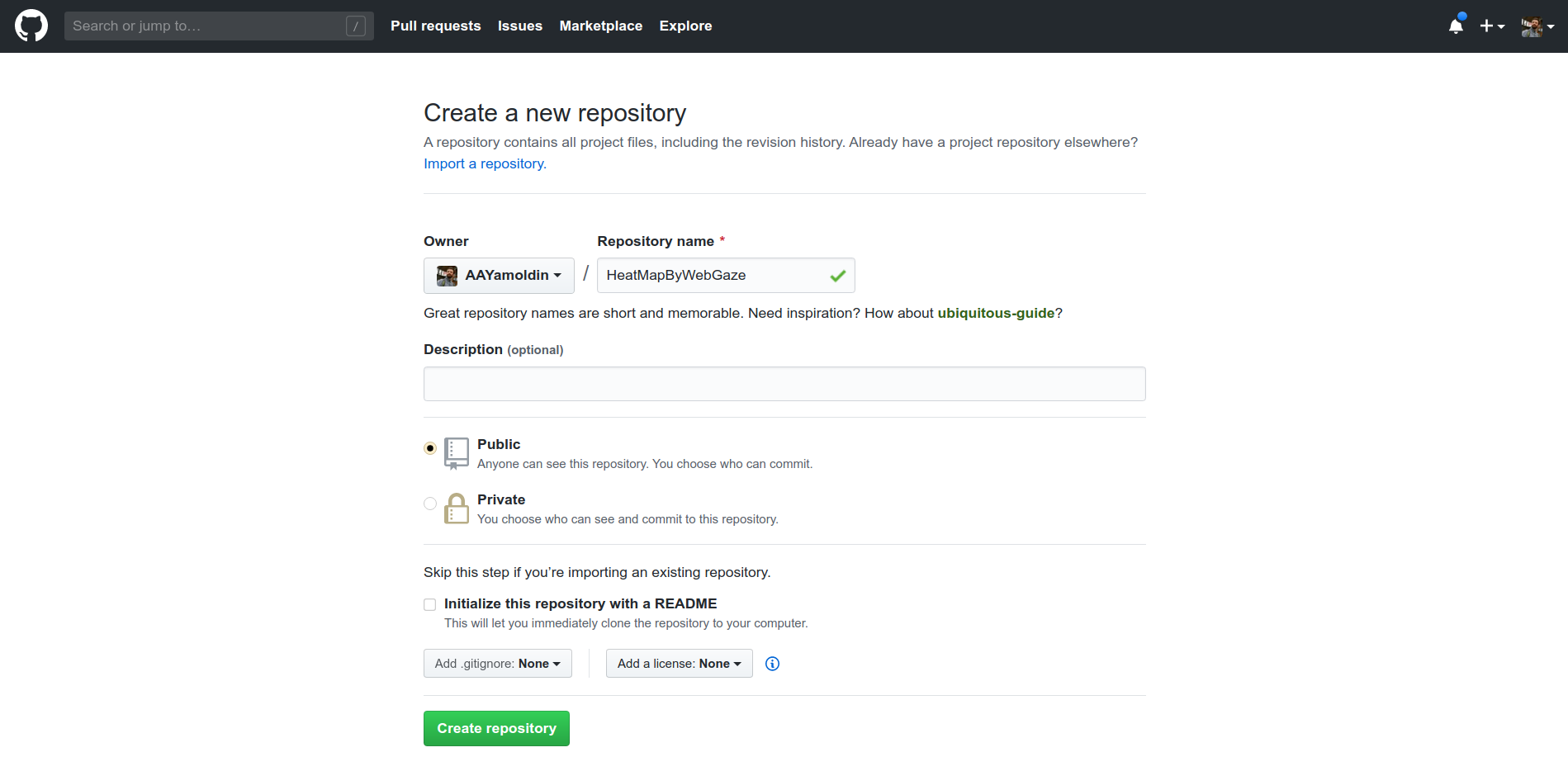


Рис.18. Создание нового репозитория GitHub

Затем необходимо склонировать репозиторий на компьютер. Для это следует перейти в директорию, в которой мы желаем хранить свой проект и из этой директории выполнить в консоли команду git clone <repo URL> в нашем случае это выглядит как показано на рис.19

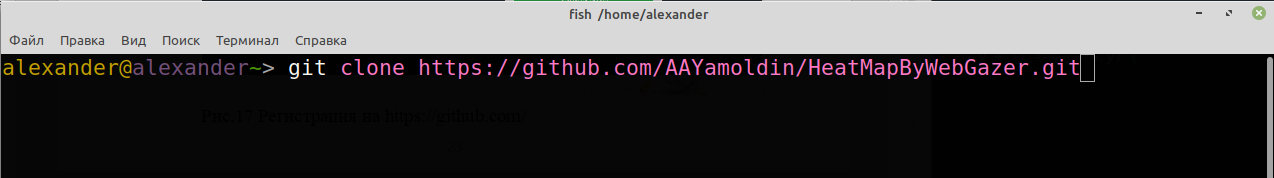


Рис.19 Клонирование репозитория в корневой каталог

После этого добавляем файлы приложения в склонированный нами репозиторий и, когда все файлы добавлены, и приложение отлажено на локальном сервере отправляем (пушим) файлы на удаленнный репозиторий GitHub(рис. 20). Для этого в каталоге репозитория нужно прописать следующие команды:

* git add -A - добавляем в индекс все новые файлы, появвшиеся в директории репозитория в индекс. Флаг -А (all) делает это для всех файлов.
* git commit -m “text of commit” - оставляем пометку с кратким объяснением того, что означает этот коммит
* git push - отправляем всё, что было закоммичено на удаленный сервер.

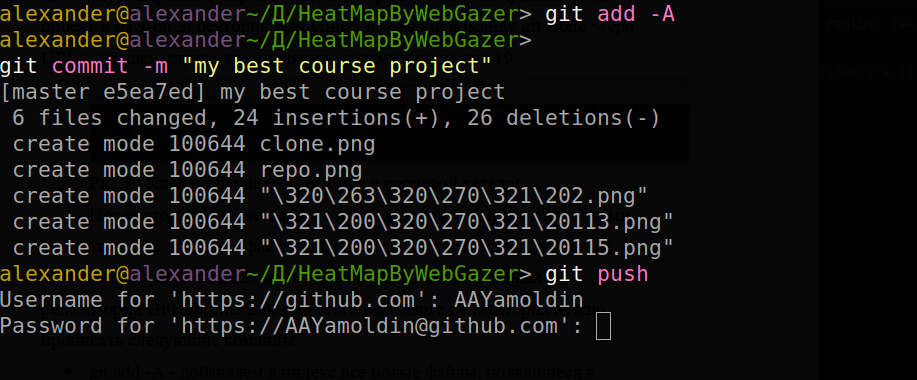


Рис. 20. Отправка коммита на удаленный репозиторий GitHub.

После этого в настройках репозитория будет показан путь к нашему приложению в сети Интернет(Рис. 21). В нашем случае это <https://aayamoldin.github.io/HeatMapByWebGazer/>

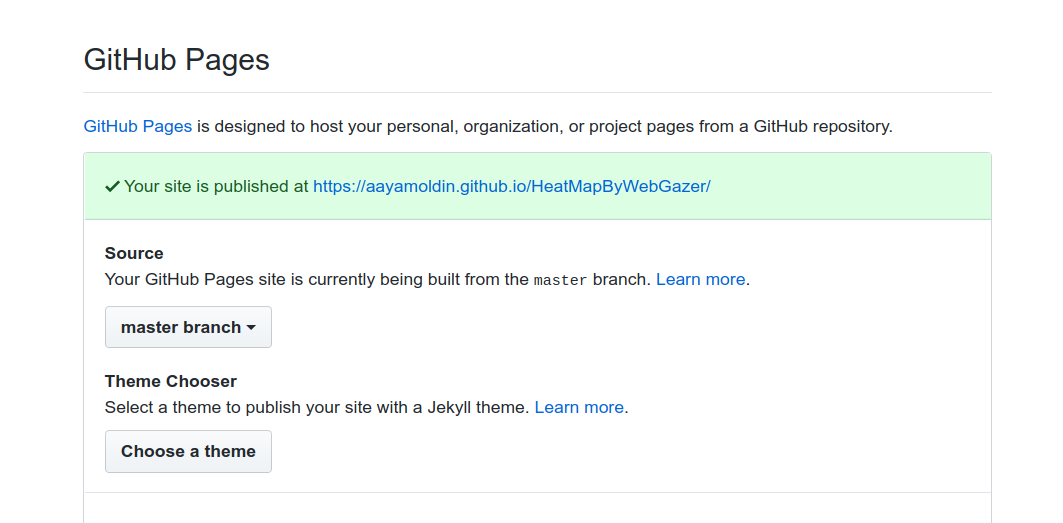


Рис.21. Путь к нашему проекту в сети Интернет.

В результате, кливнув по этой ссылке мы попадаем на страницу нашего приложения, а конкретно на страницу калибровки(рис.22), так как в главе 4.3 было сказано, что данная страница появляется по умолчанию в результате маршрутизации.

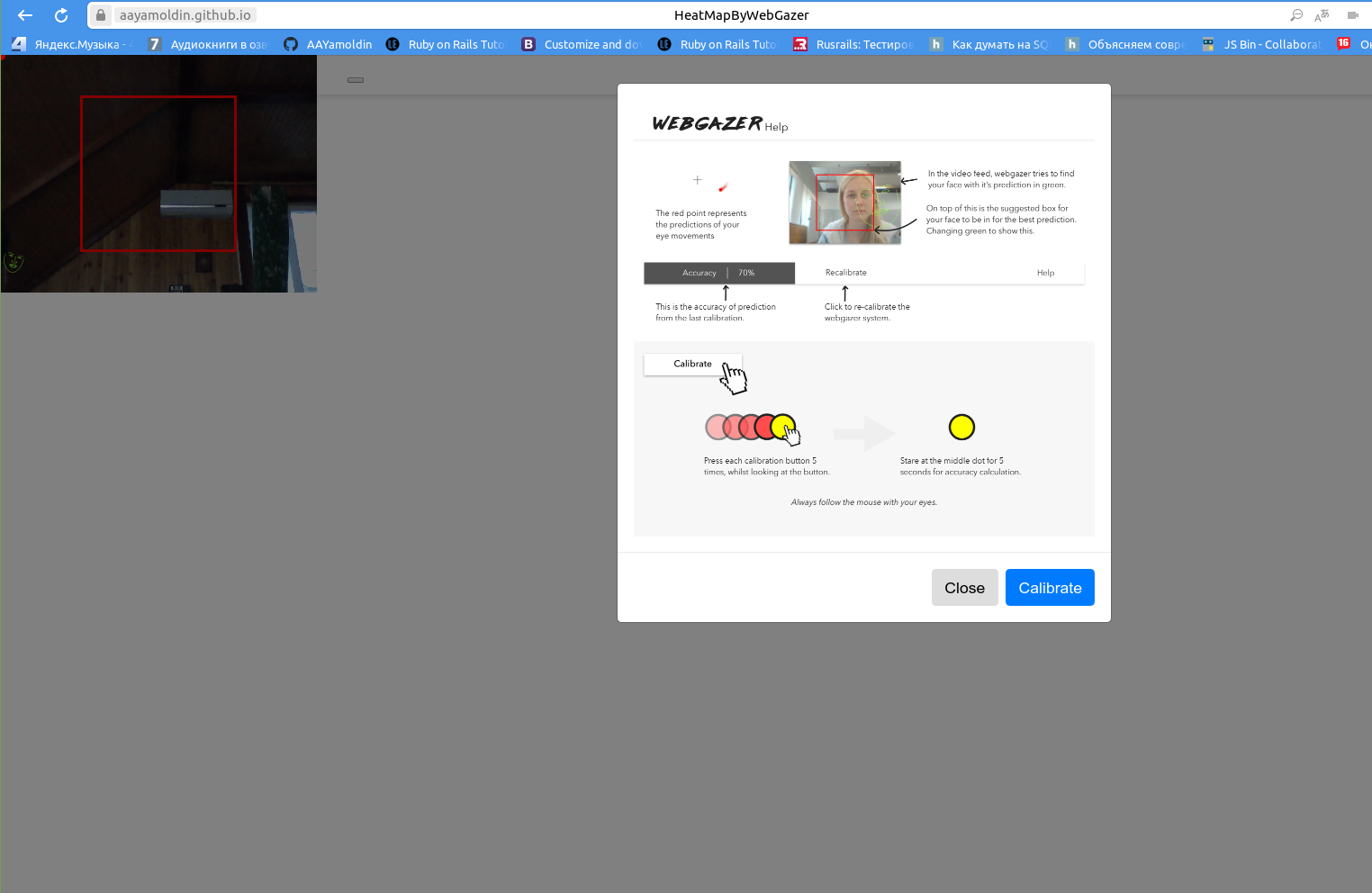


Рис. 22. AngularJS приложение в сети интернет в результате деплоя на хостинг GitHub Pages.

**Заключение**

В данной курсовой работе мы познакомились с объектно-ориентированным языком программирования JavaScript, языком гипертекстовой разметки HTML, языком, описывающим внешний вид веб документа CSS. Создали полноценное веб-прилодение с помощью фреймворка AngularJS: внутри которого использовали модуль маршрутизации, а также такие сущности как сервисы, контроллеры и представления. Внедрили внутрь приложения две JavaScript библиотеки: WebGazer.js и heatmap.js. Реализовали межскриптовую передачу данных. Дополнили функционал библиотеки heatmap.js собственной фунцией по реализации нормирования на максимальное значение. Используя крупнейший веб-сервис для IT-проектов GitHub создали собственный удаленный репозиторий, который разместили на хостинге GitHub Pages.

Конечным результатом работы стало появление в сети Интернет AngularJS приложения, отслеживающего движение глаз пользователя по экрану с помощью веб-камеры и в режиме реального времени строящего тепловую карту времени пребывания взгляда на какой-либо области экрана. Цель, поставленная в начале работы, достигнута, задачи выполнены.

**Список используемых источников**

1. Данные маркетолов [Электронный ресурс].-URL:

http://artismedia.by/blog/42-facts-about-visual-content/

2. JavaScript библиотека tensorflow [Электронный ресурс].-URL:

https://www.tensorflow.org/js/

3. JavaScript библиотека webgazer.js [Электронный ресурс].-URL:

https://webgazer.cs.brown.edu/#home

4. WebGazer: Scalable Webcam Eye Tracking Using User Interactions : Proceedings of the 25th International Joint Conference Artificial Intelligence (IJCAI) [текст] - AAAI. 2016, с. 3839 - 3845.

5. JavaScript метод getUserMedia() [Электронный ресурс].-URL:

https://developer.mozilla.org/enUS/docs/Web/API/MediaDevices/getUse rMedia

6. Браузерная поддержка JavaScript метода getUserMedia() [Электронный ресурс].-URL:

https://developer.mozilla.org/enUS/docs/Web/API/MediaDevices/getUse rMedia#Browser\_compatibility

7. Что такое фреймворк [Электронный ресурс].-URL:

https://habr.com/ru/post/476424/

8. Тело AngularJS приложения [Электронный ресурс].-URL:

https://code.angularjs.org/1.3.15/angular.min.js

9. Исходный код файла angular-route.min.js [Электронный ресурс]. URL:

https://code.angularjs.org/1.3.15/angular-route.min.js

10. Упаковка окружностей в квадрат [Электронный ресурс].-URL:

https://en.wikipedia.org/wiki/Circle\_packing\_in\_a\_square

**Приложение**

**app.js**

var app = angular.module('PredictionApp', ['ngRoute']);

app.config( function ($routeProvider) {

$routeProvider

.when('/calibration', {

controller: 'CalibrationController',

templateUrl: 'views/calibration.html'

})

.when('/workspace', {

controller: 'WorkspaceController',

templateUrl: 'views/workspace.html'

})

.otherwise({

redirectTo: '/calibration'

});

});

**CalibrationController.js**

app.controller('CalibrationController', ['$scope', 'startingWebGazer', function ($scope, startingWebGazer) {

window.onload = function () {

$scope.startingWebGazer = startingWebGazer.startgazer();

var setup = function () {

//Set up the main canvas. The main canvas is used to calibrate the webgazer.

var canvas = document.getElementById("plotting\_canvas");

canvas.width = window.innerWidth;

canvas.height = window.innerHeight;

canvas.style.position = 'fixed';

};

function checkIfReady() {

if (webgazer.isReady()) {

setup();

} else {

setTimeout(checkIfReady, 100);

}

}

setTimeout(checkIfReady, 100);

};

window.onbeforeunload = function () {

console.log('user want to left the page');

//webgazer.end(); //Uncomment if you want to save the data even if you reload the page.

window.localStorage.clear(); //Comment out if you want to save data across different sessions

}

var PointCalibrate = 0;

var CalibrationPoints = {};

/\*\*

\* Clear the canvas and the calibration button.

\*/

function ClearCanvas() {

$(".Calibration").hide();

var canvas = document.getElementById("plotting\_canvas");

try {

canvas.getContext('2d').clearRect(0, 0, canvas.width, canvas.height);

} catch {

}

}

/\*\*

\* Show the instruction of using calibration at the start up screen.

\*/

function PopUpInstruction() {

ClearCanvas();

swal({

title: "Calibration",

text: "Please click on each of the 9 points on the screen. You must click on each point 5 times till it goes yellow. This will calibrate your eye movements.",

buttons: {

cancel: false,

confirm: true

}

}).then(isConfirm => {

ShowCalibrationPoint();

});

}

/\*\*

\* Show the help instructions right at the start.

\*/

$scope.helpModalShow = function () {

$('#helpModal').modal('show');

}

/\*\*

\* Load this function when the index page starts.

\* This function listens for button clicks on the html page

\* checks that all buttons have been clicked 5 times each, and then goes on to measuring the precision

\*/

$(document).ready(function () {

ClearCanvas();

$scope.helpModalShow();

$(".Calibration").click(function () { // click event on the calibration buttons

var id = $(this).attr('id');

if (!CalibrationPoints[id]) { // initialises if not done

CalibrationPoints[id] = 0;

}

CalibrationPoints[id]++; // increments values

if (CalibrationPoints[id] == 5) { //only turn to yellow after 5 clicks

$(this).css('background-color', 'yellow');

$(this).prop('disabled', true); //disables the button

PointCalibrate++;

} else if (CalibrationPoints[id] < 5) {

//Gradually increase the opacity of calibration points when click to give some indication to user.

var opacity = 0.2 \* CalibrationPoints[id] + 0.2;

$(this).css('opacity', opacity);

}

//Show the middle calibration point after all other points have been clicked.

if (PointCalibrate == 8) {

$("#Pt5").show();

}

if (PointCalibrate >= 9) { // last point is calibrated

//using jquery to grab every element in Calibration class and hide them except the middle point.

$(".Calibration").hide();

$("#Pt5").show();

// clears the canvas

var canvas = document.getElementById("plotting\_canvas");

canvas.getContext('2d').clearRect(0, 0, canvas.width, canvas.height);

// notification for the measurement process

swal({

title: "Calculating measurement",

text: "Please don't move your mouse & stare at the middle dot for the next 5 seconds. This will allow us to calculate the accuracy of our predictions.",

closeOnEsc: false,

allowOutsideClick: false,

closeModal: true

}).then(isConfirm => {

// makes the variables true for 5 seconds & plots the points

$(document).ready(function () {

store\_points\_variable(); // start storing the prediction points

sleep(5000).then(() => {

stop\_storing\_points\_variable(); // stop storing the prediction points

var past50 = get\_points() // retrieve the stored points

var precision\_measurement = calculatePrecision(past50);

var accuracyLabel = "<a>Accuracy | " + precision\_measurement + "%</a>";

document.getElementById("Accuracy").innerHTML = accuracyLabel; // Show the accuracy in the nav bar.

swal({

title: "Your accuracy measure is " + precision\_measurement + "%",

allowOutsideClick: false,

buttons: {

cancel: "Recalibrate",

confirm: true,

}

}).then(isConfirm => {

if (isConfirm) {

window.location.href = "#/workspace";

//clear the calibration & hide the last middle button

ClearCanvas();

} else {

//use restart function to restart the calibration

ClearCalibration();

ClearCanvas();

ShowCalibrationPoint();

}

});

});

});

});

}

});

});

/\*\*

\* Show the Calibration Points

\*/

function ShowCalibrationPoint() {

$(".Calibration").show();

$("#Pt5").hide(); // initially hides the middle button

}

/\*\*

\* This function clears the calibration buttons memory

\*/

function ClearCalibration() {

window.localStorage.clear();

$(".Calibration").css('background-color', 'red');

$(".Calibration").css('opacity', 0.2);

$(".Calibration").prop('disabled', false);

CalibrationPoints = {};

PointCalibrate = 0;

}

// sleep function because java doesn't have one, sourced from http://stackoverflow.com/questions/951021/what-is-the-javascript-version-of-sleep

function sleep(time) {

return new Promise((resolve) => setTimeout(resolve, time));

}

/\*\*

\* Restart the calibration process by clearing the local storage and reseting the calibration point

\*/

$scope.Restart = function () {

document.getElementById("Accuracy").innerHTML = "<a>Not yet Calibrated</a>";

ClearCalibration();

PopUpInstruction();

}

/\*

\* Sets store\_points to true, so all the occuring prediction

\* points are stored

\*/

function store\_points\_variable() {

store\_points\_var = true;

}

/\*

\* Sets store\_points to false, so prediction points aren't

\* stored any more

\*/

function stop\_storing\_points\_variable() {

store\_points\_var = false;

}

/\*

\* Returns the stored tracker prediction points

\*/

function get\_points() {

var past50 = new Array(2);

past50[0] = xPast50;

past50[1] = yPast50;

return past50;

}

function calculatePrecision(past50Array) {

var windowHeight = $(window).height();

var windowWidth = $(window).width();

// Retrieve the last 50 gaze prediction points

var x50 = past50Array[0];

var y50 = past50Array[1];

// Calculate the position of the point the user is staring at

var staringPointX = windowWidth / 2;

var staringPointY = windowHeight / 2;

var precisionPercentages = new Array(50);

calculatePrecisionPercentages(precisionPercentages, windowHeight, x50, y50, staringPointX, staringPointY);

var precision = calculateAverage(precisionPercentages);

// Return the precision measurement as a rounded percentage

return Math.round(precision);

};

/\*

\* Calculate percentage accuracy for each prediction based on distance of

\* the prediction point from the centre point (uses the window height as

\* lower threshold 0%)

\*/

function calculatePrecisionPercentages(precisionPercentages, windowHeight, x50, y50, staringPointX, staringPointY) {

for (x = 0; x < 50; x++) {

// Calculate distance between each prediction and staring point

var xDiff = staringPointX - x50[x];

var yDiff = staringPointY - y50[x];

var distance = Math.sqrt((xDiff \* xDiff) + (yDiff \* yDiff));

// Calculate precision percentage

var halfWindowHeight = windowHeight / 2;

var precision = 0;

if (distance <= halfWindowHeight && distance > -1) {

precision = 100 - (distance / halfWindowHeight \* 100);

} else if (distance > halfWindowHeight) {

precision = 0;

} else if (distance > -1) {

precision = 100;

}

// Store the precision

precisionPercentages[x] = precision;

}

}

function calculateAverage(precisionPercentages) {

var precision = 0;

for (x = 0; x < 50; x++) {

precision += precisionPercentages[x];

}

precision = precision / 50;

return precision;

}

}])

**calibration.html**

<canvas id="plotting\_canvas" width="500" height="500" style="cursor:crosshair;"></canvas>

<!-- Calibration points -->

<div class="calibrationDiv">

<input type="button" class="Calibration" id="Pt1"></input>

<input type="button" class="Calibration" id="Pt2"></input>

<input type="button" class="Calibration" id="Pt3"></input>

<input type="button" class="Calibration" id="Pt4"></input>

<input type="button" class="Calibration" id="Pt5"></input>

<input type="button" class="Calibration" id="Pt6"></input>

<input type="button" class="Calibration" id="Pt7"></input>

<input type="button" class="Calibration" id="Pt8"></input>

<input type="button" class="Calibration" id="Pt9"></input>

</div>

<!-- Modal -->

<div id="helpModal" class="modal fade" role="dialog">

<div class="modal-dialog">

<!-- Modal content-->

<div class="modal-content">

<div class="modal-body">

<img src="./media/calibration.png" width="100%" height="100%" alt="webgazer demo instructions">

</div>

<div class="modal-footer">

<button id="closeBtn" type="button" class="btn btn-default" data-dismiss="modal">Close</button>

<button type="button" class="btn btn-primary" data-dismiss="modal" ng-click="Restart()">Calibrate</button>

</div>

</div>

</div>

</div>

**WorkspaceController.js**

app.controller('WorkspaceController', ['$scope', function ($scope) {

let datapoints = [];

let max = 0;

var config = {

container: document.querySelector('.heatmap'),

radius: 50,

};

var heatmapInstance = h337.create(config);

webgazer.setGazeListener(function (data, elapsedTime) {

let valpoint = 0;

if (data == null) {

return;

}

try {

let elem = document.querySelector('.heatmap').getBoundingClientRect()

datapoints.push({

x: (data.x - elem.left),//these x coordinates are relative to the viewport

y: (data.y - elem.top),//these y coordinates are relative to the viewport

value: 16

})

datapoints.forEach((item) => {

for (let i = -config.radius; i < config.radius; i++) {

for (let j = -config.radius; j < config.radius; j++) {

try {

if ((item.x + i) === data.x && (item.y + j) === data.y) {

valpoint = valpoint + 1;

}

} catch (e) {

console.log(e);

}

}

}

})

max = Math.max(max, valpoint);

let heatmap = {

max: max,

min: 0,

data: datapoints

};

heatmapInstance.setData(heatmap);

} catch (e) {

console.log(e)

}

});

}]);

**workspace.html**

<div class="demo-wrapper">

<h1 class="zone">The most intresting zone!</h1>

<div class="heatmap"></div>

</div>

**startingWebGazer.js**

app.factory('startingWebGazer', function () {

return {

startgazer: function () {

webgazer.begin()

.showPredictionPoints(true);

}}

});

**index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>HeatMapByWebGazer</title>

<!-- styles files -->

<link rel="stylesheet" type="text/css" href="css/style.css">

<link rel="stylesheet" href="css/bootstrap.min.css">

<!-- the most important scripts -->

<script async src="./js/src/webgazer.js"></script>

<script src="./js/vendor/sweetalert.min.js"></script>

<script src="./js/vendor/jquery.min.js"></script>

<!-- AngularJS -->

<script src="./js/vendor/angular.min.js"></script>

</head>

<nav id="webgazerNavbar" class="navbar navbar-default navbar-fixed-top">

<div class="container-fluid">

<div class="navbar-header">

<!-- The hamburger menu button -->

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

</div>

<div class="collapse navbar-collapse" id="myNavbar">

<ul class="nav navbar-nav">

<!-- Accuracy -->

<li id="Accuracy"><a>Not yet Calibrated</a></li>

<li ng-click="helpModalShow()"><a href="#/calibration">Recalibrate</a></li>

</ul>

</div>

</div>

</nav>

<body ng-app="PredictionApp">

<div ng-view></div>

<!-- Modules -->

<script src="./js/app.js"></script>

<!-- Controllers -->

<script src="./js/controllers/CalibrationController.js"></script>

<script src="./js/controllers/WorkspaceController.js"></script>

<!-- services -->

<script src="./js/services/startingWebGazer.js"></script>

<script src="./js/services/heatmap.js"></script>

<script src="./js/vendor/angular-route.min.js"></script>

<!-- Other scripts -->

<script src="./js/vendor/bootstrap.min.js"></script>

</body>

</html>

**style.css**

body {

min-width:600px!important;

}

/\* Video Feed Styling \*/

#overlay {

z-index: 1;

}

/\* Navbar styling \*/

#webgazerNavbar {

left: 320px;

z-index: 0;

box-shadow: 0px 3px 5px #ddd;

border:0px;

background-color: #fff;

}

#Accuracy a{

background-color: #222;

color: #eee;

left: -15px;

padding-left: 80px;

padding-right: 40px;

}

#Accuracy {

background-color: #222;

padding-left: inherit;

}

li {

padding-left: 50px;

}

.navbar-right li {

padding-right: 20px;

}

.helpBtn {

padding: 15px;

border: none;

background: none;

color: #777;

}

.helpBtn a {

background: none;

color: #777;

}

.helpBtn a:hover {

color: black!important;

text-decoration: none;

}

.modal-header {

border-bottom: none!important;

}

/\* Calibration button styling \*/

.Calibration{

width: 20px;

height: 20px;

-webkit-border-radius: 25px;

-moz-border-radius: 25px;

border-radius: 25px;

background-color: red;

opacity: 0.2;

border-color: black;

border-style: solid;

position:fixed;

}

/\* Calibration point position \*/

#Pt1{

top: 70px;

left:340px;

}

#Pt2{

top: 70px;

left:50vw;

margin-left: 340px;

}

#Pt3{

top: 70px;

right:2vw;

}

#Pt4{

top:50vh;

left:2vw;

}

#Pt5{

top: 50vh;

left: 50vw;

}

#Pt6{

top: 50vh;

right:2vw;

}

#Pt7{

bottom:2vw;

left: 2vw;

}

#Pt8{

bottom:2vw;

left:50vw;

}

#Pt9{

bottom:2vw;

right:2vw;

}

.demo-wrapper {

height:1700px;

background:rgba(0,0,0,.03);

border:3px solid black;

width: 100%;

min-width: 600px;

position: relative;

}

.heatmap {

position: absolute;

width:100%;

height:100%;

}

.zone {

margin-left: 320px;

background-color: red;

}