|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
| Берсенёв Илья Иванович  **Разработка системы диагностики артроза по рентген снимкам коленного сустава**  *Курсовой проект*  студента образовательной программы «Программная инженерия» по направлению подготовки *09.03.04 Программная инженерия*  Руководитель,  приглашенный преподаватель  М.Д. Чистогов |

Пермь, 2024 год

**Оглавление**

[Оглавление 2](#_Toc130676825)

[Введение 3](#_Toc130676826)

[1 Анализ предметной области 4](#_Toc130676827)

[1.1 Анализ назначения программы 5](#_Toc130676828)

[1.2. Обзор существующих решений 6](#_Toc130676829)

[1.3. Анализ средств 7](#_Toc130676830)

[1.3.1. Сервисы для геоданных 7](#_Toc130676831)

[1.3.2. Графические сервисы 8](#_Toc130676832)

[1.4. Требования к проекту 9](#_Toc130676833)

[1.5. Диаграмма прецедентов 10](#_Toc130676834)

[2 Проектирование 11](#_Toc130676835)

[2.1 Общая информация 11](#_Toc130676836)

[2.2 Проектирование интерфейса пользователя 12](#_Toc130676837)

[2.2.1 Экран карты 13](#_Toc130676838)

[2.2.2. Экран флагов 14](#_Toc130676839)

[2.2.3 Экран формы 15](#_Toc130676840)

[2.2.4. Общий экран 17](#_Toc130676841)

[2.3 Проектирование данных 19](#_Toc130676842)

[2.3.1. Проектирование базы данных 19](#_Toc130676843)

[2.3.2. Проектирование запросов к базе данных 21](#_Toc130676844)

[2.3.3. Результаты проектирования 22](#_Toc130676845)

[3 Реализация информационной системы 24](#_Toc130676846)

[3.1. Архитектура 25](#_Toc130676847)

[3.1.1. Модуль utils.py 25](#_Toc130676848)

[3.1.2. Модуль app.py 26](#_Toc130676849)

[3.1.3. Шаблон test.html 27](#_Toc130676850)

[3.1.3.1. Создание слоя из переданного JSON объекта 27](#_Toc130676851)

[3.1.3.2. Создание слоя из переданного JSON объекта 28](#_Toc130676852)

[3.1.3.3. Создание формы 28](#_Toc130676853)

[3.2. Финальный вид приложения 30](#_Toc130676854)

[Заключение 31](#_Toc130676855)

[Список литературы 32](#_Toc130676856)

# Введение

В наше время наблюдается стремительный рост технологий с применением искусственного интеллекта, под которым часто подразумевают нейронные сети. Во многом это обусловлено тем, что нейронные сети, будучи симуляцией нейронов органического мозга, позволяют решать нетривиальные задачи, неподвластные человеку, например постановка диагноза по рентген снимку, предсказание курса акций на бирже, генерация изображений и т.д.

Ввиду востребованности данной технологии была выбрана тема «разработка системы диагностики артроза по рентген снимкам коленного сустава».

# Анализ предметной области

в данной главе осуществляется анализ предметной области, перечисляются существующие решения, а также ведётся обзор технологий для реализации сервиса.

**1.1 Анализ назначения программы**

В наше время нейросетевые технологии используются во множестве различных сфер, однако по большей части они решают задачи бизнеса, а разработок в области медицины на порядок меньше.

Сервис будет представлять из себя систему, которую можно будет интегрировать в систему ЕРИС, способную распознавать степень артроза сустава по его рентген снимку, однако хотелось бы подчеркнуть, что на первом месте стоит именно функциональность сервиса, а на втором – его способность к интеграции.

Сервис будет реализовывать единственную функцию:

1. Загрузка рентген снимка сустава и мгновенный ответ с вероятностями степеней артроза в соответствии с требованиями ЕРИС.

**1.2. Обзор существующих решений**

На момент написания курсовой работы не было выявлено аналогов данной работы. Существуют предобученные нейронные сети, но они имеют определённые недостатки: несовместимы с ЕРИС, не имеют возможности подсветки патологии либо имеют слишком низкую точность.

**1.3. Анализ средств**

Сервис для распознавания степени артроза будет иметь в себе следующие компоненты:

* Ядро, представляющее из себя нейросеть;
* Веб-сервис, служащий обёрткой над нейросетью;
* Брокер сообщений для интеграции с ЕРИС.

Были поставлены следующие задачи:

1. Изучить сервисы/технологии для разработки нейросетей;
2. Изучить подходы к разработке нейросетей;
3. Изучить сервисы, предоставляющие ресурсы для обучения нейросетей;
4. Выбрать веб-сервис для обертки над нейросетью;
5. Выбрать брокер сообщений;
6. Сформировать требования к продукту;
7. Спроектировать приложение.

**1.3.1. Сервисы для разработки нейросетей**

В данный момент в сфере нейросетей присутствуют три лидера: TensorFlow, PyTorch, Keras и XGBoost. После непродолжительного анализа был выбран фреймворк Keras ввиду того, что в нём присутствует возможность составлять нейросети высокоуровневыми блоками, что довольно удобно.

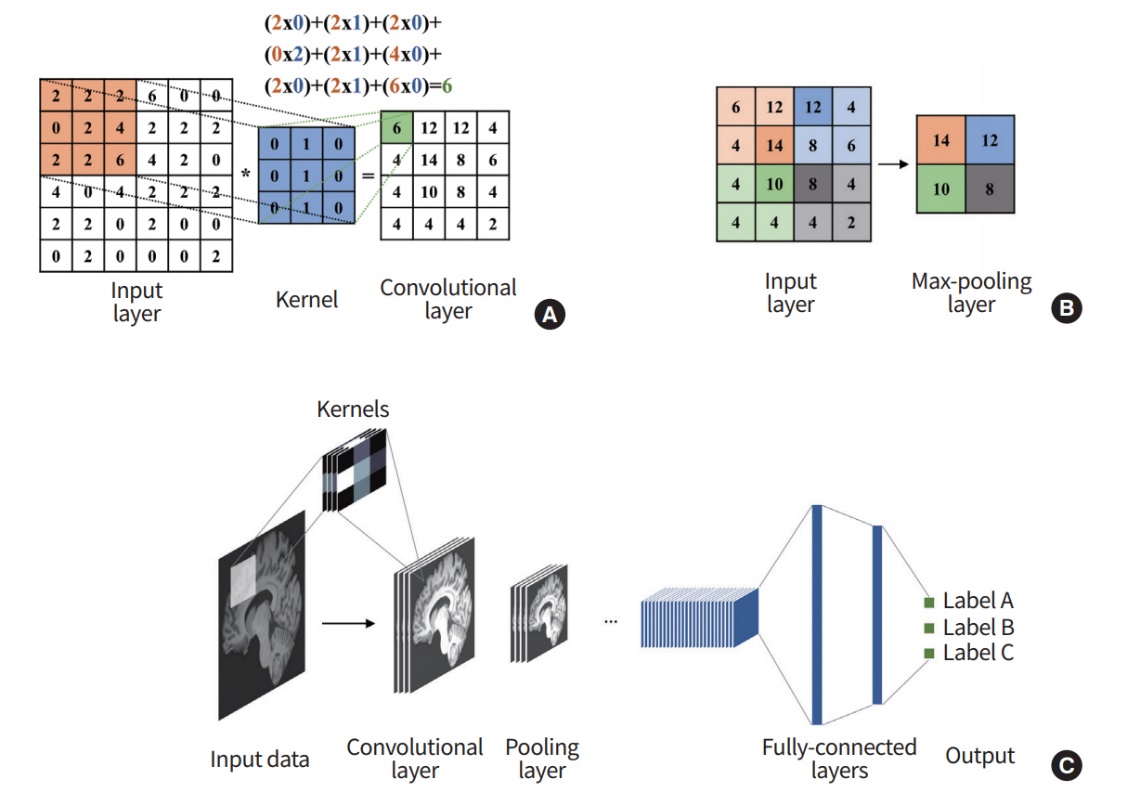
**1.3.2. Подходы к разработке нейросетей**

В процессе самообучения было выделено несколько различных подходов к разработке нейросетей, а именно:

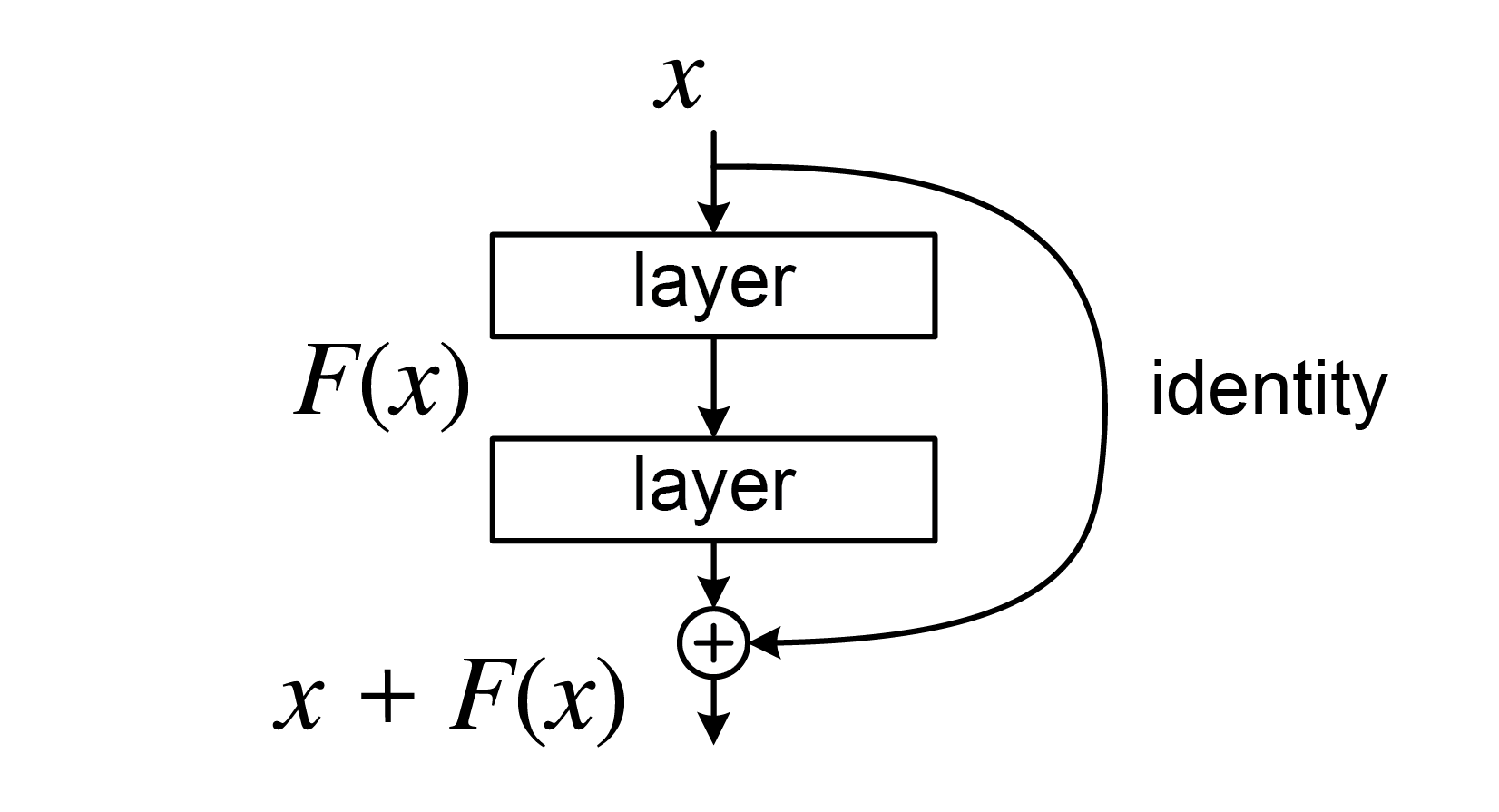
1. Свёрточные нейронные сети (convolutional neural networks);
2. Остаточные нейронные сети (residual neural networks);
3. Рекуррентные нейронные сети (recurrent neural networks).

Рассмотрим каждый из них поподробнее.

Свёрточные нейронные сети представляют из себя архитектуру нейронных сетей, при которой на вход нейронной сети подаётся определённое количество сигналов, которые сеть преобразует в некоторое количество feature maps. Feature map можно охарактеризовать как исходные данные с наложенным «фильтром». На рис. 1 сверху слева вы можете видеть вычисление feature map из исходных данных и сверточного ядра. Как мы можем видеть, это обыкновенное скалярное произведение входных данных со сверточным ядром с тем лишь нюансом, что скалярное вычисление вычисляется для каждой области входных данных. Это значит, что если мы условно обозначим на картинке первую область как [0, 2;0, 2], где 0,2 это координаты начала и конца области по вертикали и горизонтали соответственно, то также будут вычислены [0, 2; 1, 3], [0, 2; 2, 4], [0, 2; 3, 5], [1, 3; 0, 2] и так далее. Данный вид архитектуры высокоэффективен для нахождения повторяющихся закономерностей.



Остаточные нейронные сети являются типом нейросетей, в которых имеются особые остаточные блоки, в которых могут быть, например, линейные или сверточные слои, которые так же обращаются ко входу остаточного блока. Происходит это посредством суммирования обработанного каким-либо слоём и необработанного входов блока. Вы можете увидеть примерную архитектуру такого блока на рисунке 2. Данный вид архитектуры эффективен тем, что теряется меньше информации в процессе преобразования данных, чем в линейной или сверточной нейросетях.



Рекуррентные нейронные сети – это вид нейронных сетей, имеющих долгую краткосрочную память (LSTM). Чаще всего эти нейросети используются в языковых моделях, так что разбирать их не имеет особого смысла.

Таким образом, для разработки были выбраны 2 основных архитектуры: свёрточные нейронные сети и остаточные нейронные сети.

**1.3.3. Сервисы для обучения нейросети**

Ввиду того, что обучение нейросети – довольно сложная задача в смысле компьютерных вычислений, необходимо было выбрать бесплатный сервис для обучения. Наиболее популярные – Google Colab, Kaggle, Microsoft Azure Notebook. Ввиду того, что из-за санкций последний вариант недоступен, пришлось пользоваться Colab и Kaggle. Colab довольно удобен с точки зрения разработки, поэтому изначально работа велась в нём, однако Kaggle предоставляет бóльшие ресурсы для тренировки нейросетей, потому дальнейшая разработка велась в Kaggle.

**1.3.4. Сервис-обёртка над нейросетью**

Для совместимости с ЕРИС необходимо, чтобы сервис был размещён в интернете, а для этого нужен веб-фреймворк. В качестве такого был выбран FastApi ввиду того, что это активно развивающийся и легкий фреймворк.

**1.3.5. Брокер сообщений**

Для совместимости с ЕРИС также необходим брокер сообщений Kafka. Именно этот брокер был выбран для системы, для взаимодействия с ним был выбран пакет aiokafka.

**1.4. Требования к проекту**

Были сформированы следующие требования к проекту:

1. Проект должен распознавать степень артроза сустава;
2. Проект должен возвращать области, в которых предположительно локализованы патологии;
3. Проект должен быть совместим с ЕРИС.

# Проектирование

в данной главе будет производиться проектирование самого сервиса и его пользовательского интерфейса.

**2.1 Общая информация**

При проектировании было принято решение сделать конечный продукт максимально простым. Для этого было решено отдавать предпочтение готовым технологиям вместо самописных, таким как BootStrap для разметки страниц, ArcGis для отображения карты, Pandas для работыс геоданными, Flask для серверной архитектуры. Для работы сервиса будут использоваться 2 CSV таблицы: одна – для маркеров, вторая – для всех данных. Нет нужды их описывать, т.к. в обеих более 100 полей.

**2.2 Проектирование интерфейса пользователя**

Перед началом работы было необходимо сверстать или придумать пользовательский интерфейс, который затем обретёт программа, являющаяся продуктом курсовой деятельности. Было принято решение рисовать эскизы в графическом редакторе под названием «Microsoft Paint», ввиду его лёгкости и быстроты.

В конечном продукте, результате курсовой работы будет всего лишь один экран с тремя секциями. Так будет из-за того, что приложение планируется сделать одностраничным, с минимальной перегруженностью и максимальной полезностью.

На главном и единственном экране будут три элемента: интерактивная карта, реализованная средствами ArcGis, список из флагов специальностей, а также форма для добавления новой клиники.

Интерфейс самой программы будет реализован на веб-платформе, что значит, что он будет оформляться с помощью веб-фреймворков. Был выбран фреймворк Bootstrap, призванный облегчать и максимально упрощать работу с CSS и HTML. Он был выбран ввиду его простоты и гибкости, что идеально для небольших проектов.

**2.2.1 Экран карты**

Учитывая, что для отображения географической информации был выбран такой JavaScript-фреймворк как ArcGis API, тепловая карта будет выглядеть именно такой, какой она представлена на сайте с документацией с минимумом косметических модификаций. Внешний вид тепловой карты представлен на рисунке 2.

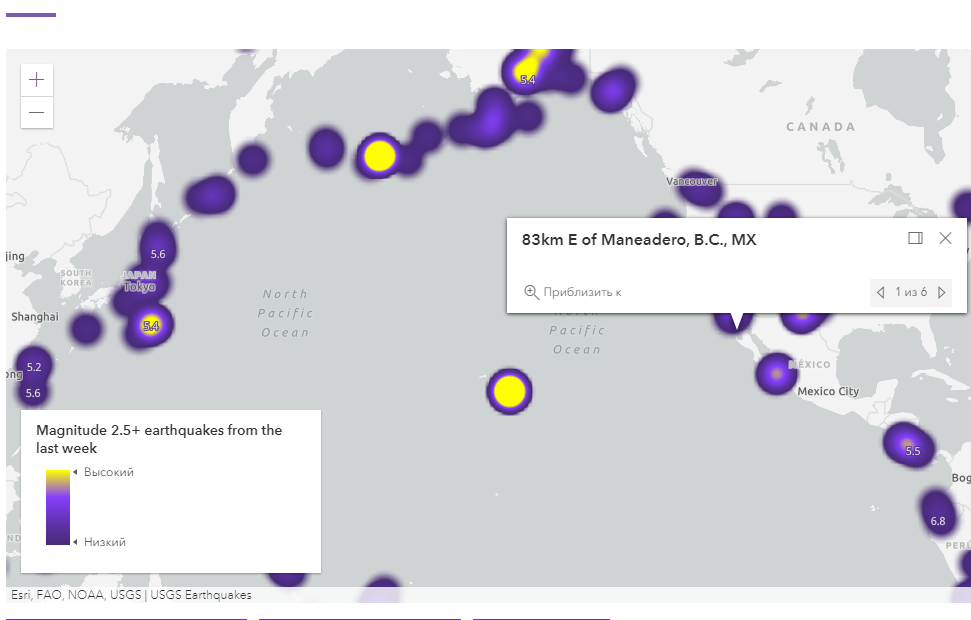


Рисунок 2 – вид карты

В данном дизайн-макете представлены 4 основных элемента.

Первый из них – это легенда в нижнем левом углу. Легенда полуавтоматически генерируется фреймворком и нужна для того, чтобы обозначать уровень концентрации элементов в одной позиции.

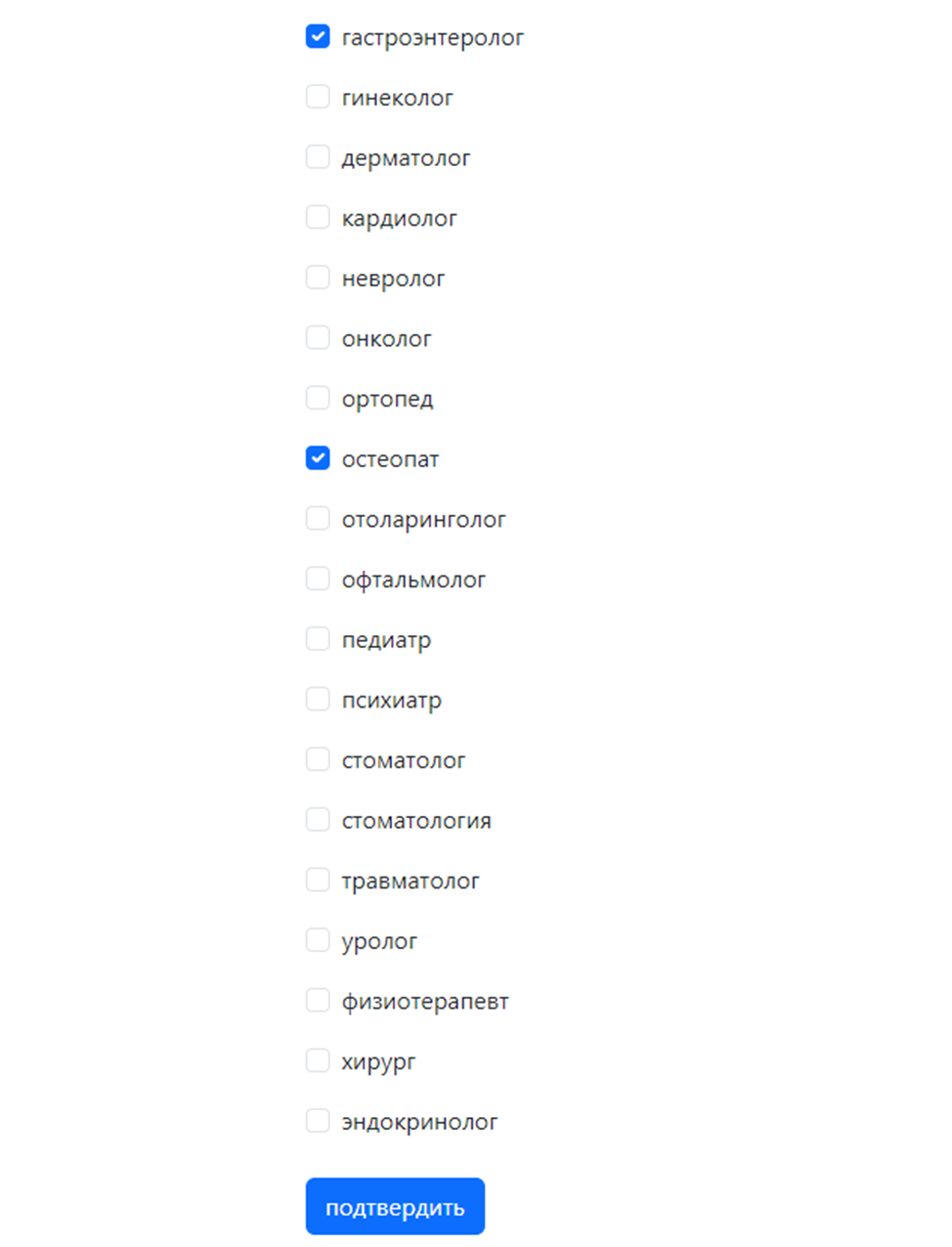
Второй элемент – это задний план. Задний план полуавтоматически генерируется фреймворком и представляет из себя векторный слой, изображающий земную карту.

Третий элемент – непосредственно тепловая карта. Ее уже нужно задавать явно, и для этого будет использоваться конвертирование геообъектов из csv в json с дальнейшей передачей в шаблон страницы через инструмент jinja из фреймворка Flask.

Четвёртый и последний элемент – всплывающее окно. Его можно настраивать под себя, но кастомизация очень ограниченная: в основную часть можно добавлять лишь текст, а в заголовок – кнопки. В конечном сервисе всплывающее окно будет иметь название клиники в заголовке, информацию о ней в основной части, а кнопки будут отвечать за маркировку и редактирование.

**2.2.2. Экран флагов**

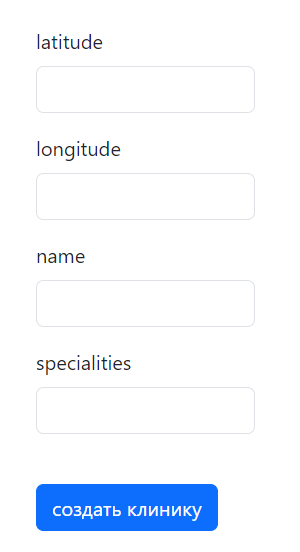
В приложении необходимо реализовать фильтрацию по специальности, а потому для этого было принято решение создать экран с одними лишь флагами и кнопкой «подтвердить». Это решение было принято в связи с концепцией минимальной трудоёмкости, так как данную идею реализовать проще всего. При нажатии на кнопку «подтвердить» форма будет посылать пост-запрос на главную страницу сервиса, после чего будет возвращаться страница с уже отфильтрованной по специальностям информацией. Примерный внешний вид изображён на рисунке 3.



***Рисунок 3 – внешний вид флагов***

**2.2.3 Экран формы**

Учитывая концепцию минимальной трудоёмкости, было принято решение реализовать форму в виде минимума полей и одной кнопки. Поля – это два элемента ввода чисел (долгота и широта), а также 2 элемента ввода текста (название частной профильной клиники и специальности частной профильной клиники). Кнопка «создать клинику» завершает создание формы и посылает данные из формы на другую страницу сайта. Прототип изображён на рисунке 4.



***Рисунок 4 – внешний вид формы***

**2.2.4. Общий экран**

В результате проектирования было принято решение о том, чтобы уделить основное место карте, так как она является самым важным объектом на сайте, а рядом с ней разместить флаги и кнопку подтверждения. Снизу была размещена форма добавления клиники. Таким образом, дизайн-макет на данный момент выглядит как на рисунке 5.

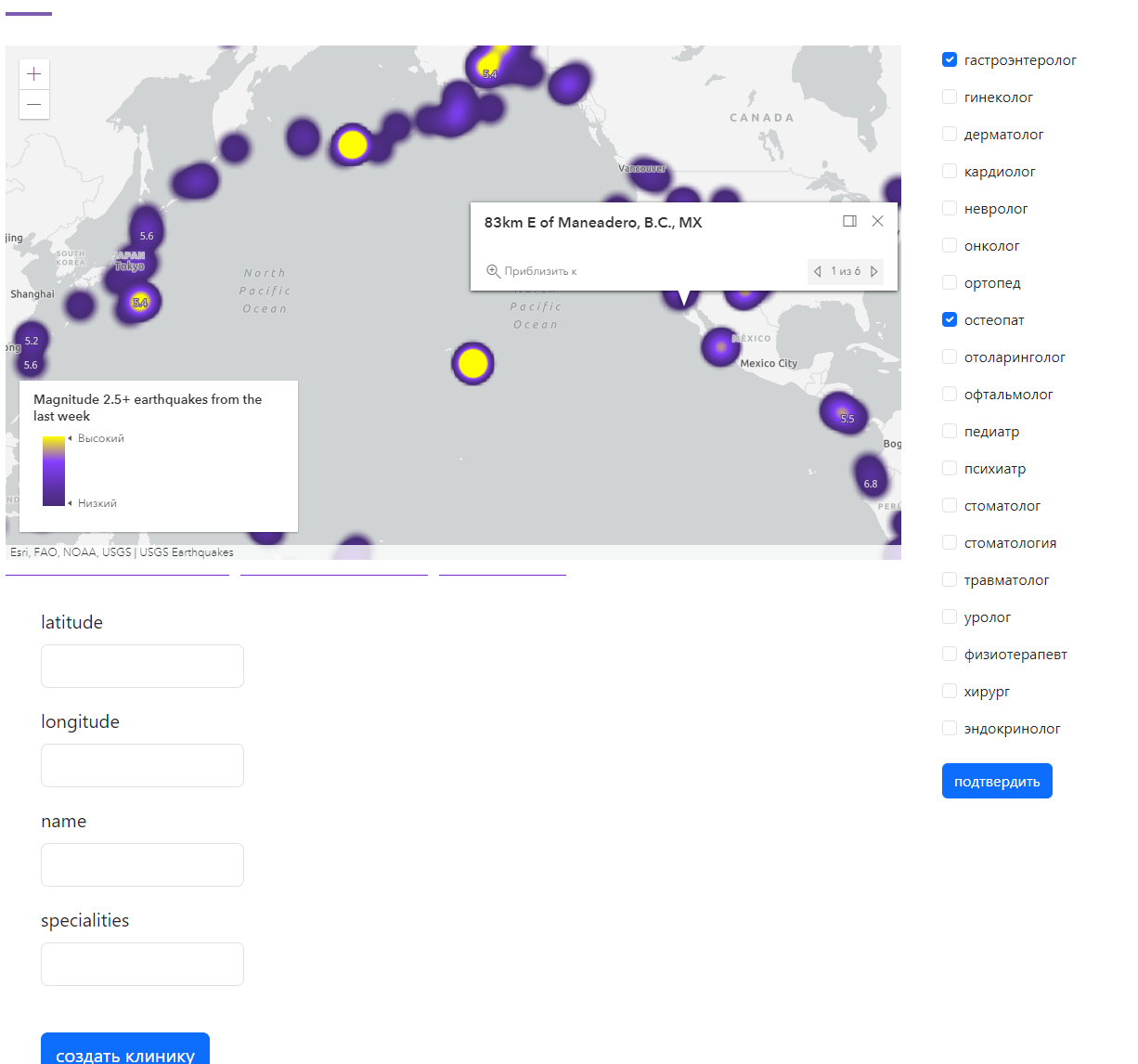


Рисунок 5 – общий макет

В данном интерфейсе невозможно совершить инъекцию с помощью редактирования и инъекцию с помощью добавления.

Во время редактирования данных о клинике будет происходить следующее: будет запрашиваться уникальный для каждого геообъекта параметр osmid, а далее по этому параметру будут происходить изменения с помощью библиотеки Pandas. Инъекция невозможна по причине того, что фреймворк ArcGis реализован так, что абсолютно все объекты на карте, реализованной с помощью JS-модуля ArcGis, имеют свой уникальный параметр osmid, и невозможно выделить элемент, который не имеет параметра osmid либо элемент, которого нет в базе данных в формате csv, потому что абсолютно все записи из базы данных отображаются на интерактивной карте.

Инъекция с помощью добавления же невозможна потому, что osmid для новой записи генерируется на сервере, притом случайно, а все поля, которые пользователь может заполнить, имеют тип string, из-за чего что бы пользователь не писал, любой контент, написанный пользователем, может находиться в базе данных.

Обработка исключительных ситуаций реализована средствами Flask: при возникновении ошибки на серверной стороне Flask выдаёт вместо запрашиваемой страницы пустую страницу с кодом 500.

Данные в систему вводятся только вручную, потому что для того, чтобы пользователь мог импортировать данные из CSV, ему необходимо создать большую CSV таблицу на несколько десятков полей.

**2.3 Проектирование данных**

Учитывая, что в моей работе не использовались базы данных в традиционном понимании, такие как MySQL, PostgreSQL, Microsoft SQL Server, Oracle DB, Mongo DB и прочие фреймворки и диалекты для работы с базами данных, будут приведены функции, написанные для работы с CSV- таблицами, поскольку CSV-таблицы выполняют роль базы данных, а функции, работающие с ними – роль запросов к базам данных.

**2.3.1. Проектирование базы данных**

Учитывая, что для получения информации я использую API OpenStreetMap, который возвращает данные в формате Geopandas.Geodataframe, который впоследствии явно приводится к типу Pandas.Dataframe и сохраняется в CSV, будет описана ровно та таблица, которая получена таким путём, притом будут описаны не все поля, а лишь самые основные ввиду их большого количества. Данная база данных не будет нормализовываться ввиду того, что Pandas не предоставляет механизмы ключей и вследствие этого приведения к нормальным формам будут бессмысленны. Ввиду особенностей Pandas был опущен столбец значение по умолчанию, так как значение по умолчанию – всегда nan. Также из-за особенностей Pandas были опущены такие столбцы как обязательность и ограничения. Получившиеся данные были описаны в таблице 1.

Таблица 1 – данные в таблице

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип данных | комментарий |
| element\_type | Строка | Строка, означающая тип геообъекта. Всегда равна node |
| osmid | Число | Уникальный идентификатор геообъекта |
| Amenity | Строка | Тип геообъекта здравоохранения. |
| contact:phone | Строка | Контактный телефон |
| contact:website | Строка | Вебсайт |
| healthcare | Строка | Тип геообъекта здравоохранения. Устаревший |
| name | Строка | Название геообъекта |
| opening\_hours | Datetime.time | Поле «часы открытия» связано с встроенной библиотекой Datetime |
| geometry | Shapely.Point | Поле «геометрия» представляет из себя класс библиотеки Shapely. Исходные данные содержали также классы Polygon и Way, однако все они были преобразованы в центроид Point. |
| alt\_name | Строка | Строка, означающая дополнительное имя клиники |
| Latitude | Дробное | Ширина |
| Longitude | Дробное | Долгота |
| addr:city | Строка | Город, в котором расположена клиника |
| addr:country | Строка | Страна, в которой расположена клиника |
| Description | Строка | Описание клиники |
| Postal\_code | Строка | Почтовый индекс клиники |
| Addr:street | Строка | Название улицы, на которой расположена клиника |
| Addr:housenumber | Число | Номер дома, в котором расположена клиника |

**2.3.2. Проектирование запросов к базе данных**

Первый запрос – edit\_csv. Он реализован с помощью библиотеки pandas и позволяет модифицировать строку в CSV-таблице, основываясь на её osmid. Если транслировать этот запрос в SQL, то получится нечто вроде кода, приведённого ниже. Сам код представлен на рисунке 6.

UPDATE csv

SET csv.name=name,

Csv.healthcare:specialty = specialities

WHERE csv.osmid = osmid;

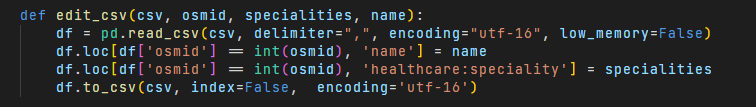


Рисунок 6 – функция edit csv

Второй запрос – mark\_csv. Он тоже реализован с помощью pandas и нужен для того, чтобы добавлять строки с определённым osmid из исходной таблицы в таблицу маркеров. Сам код представлен на рисунке 7. При трансляции на язык SQL получится следующий запрос:

INSERT INTO csv VALUES (SELECT \* FROM source WHERE source.osmid=osmid);

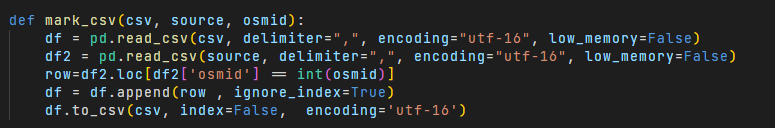


Рисунок 7 – функция mark csv

Третий запрос – mini\_df. Дело в том, что при загрузке данных в веб-страницу, чтобы фреймворк ArcGis мог их прочитать, используется обыкновенный json, и если приводить к этому формату исходную таблицу, то страницы загружаются крайне долго. Поэтому было принято решение о минификации таблиц перед их отправкой в таблицу, что и реализовано в данном запросе. Сам код представлен на рисунке 8. При трансляции в SQL получится следующий код:

SELECT geometry, osmid, name, addr:street, addr:housenumber, contact:website, contact:phone, healthcare:speciality, latitude, longitude FROM df;

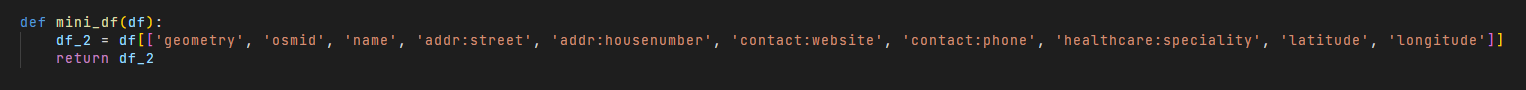


Рисунок 8 – mini df

Четвёртый запрос – add\_csv. Данный запрос позволяет добавлять в CSV-таблицу новую строку. Данный запрос необходим для того, чтобы создавать новые клиники. Сам код представлен на рисунке 9. При трансляции в SQL получится следующий код:

INSERT INTO csv (element\_type, geometry, osmid, latitude, longitude, name, healthcare:speciality) VALUES (“node”, Point(latitude, longitude), osmid, latitude, longitude, name, healthcare:speciality);

В данном запросе можно заметить дублирование данных. Это сделано для корректной работы ArcGis, так как в некоторых модулях ArcGis требуются экземпляры класса Point, а в некоторых – параметры latitude и longitude.

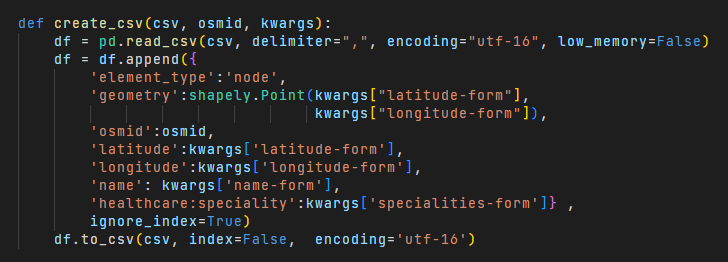


Рисунок 9 – create csv

**2.3.3. Результаты проектирования**

В ходе проектирования были получены:

1. Модель бд;
2. Модель пользовательского интерфейса;
3. Запросы к бд.

полученные результаты будут использованы в реализации системы.

# Реализация информационной системы

В данной главе будет описано получившееся приложение, а также его архитектура.

**3.1. Архитектура**

На финальной стадии разработки структура приложения выглядит так.

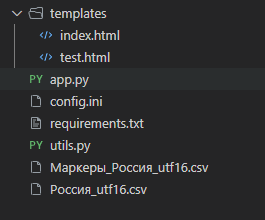


Рисунок 10 – архитектура

Templates в данной иерархии – папка с HTML шаблонами для фреймворка Flask. App.py – главный модуль, который открывает порт на ip 0.0.0.0, а также производит некоторые манипуляции с геоданными перед их передачей в шаблон.

Utils.py – это модуль, в котором содержатся «запросы» к таблицам. Он импортируется из главного модуля.

Маркеры\_Россия\_utf16.csv – изначально пустая таблица, записи из которой используются для расстановки маркеров.

Россия\_utf16.csv – таблица с данными о всех известных частных профильных клиниках в России.

Рассмотрим каждый модуль подробнее.

**3.1.1. Модуль utils.py**

Модуль utils.py представляет из себя небольшую библиотеку для работы с геоданными клиник

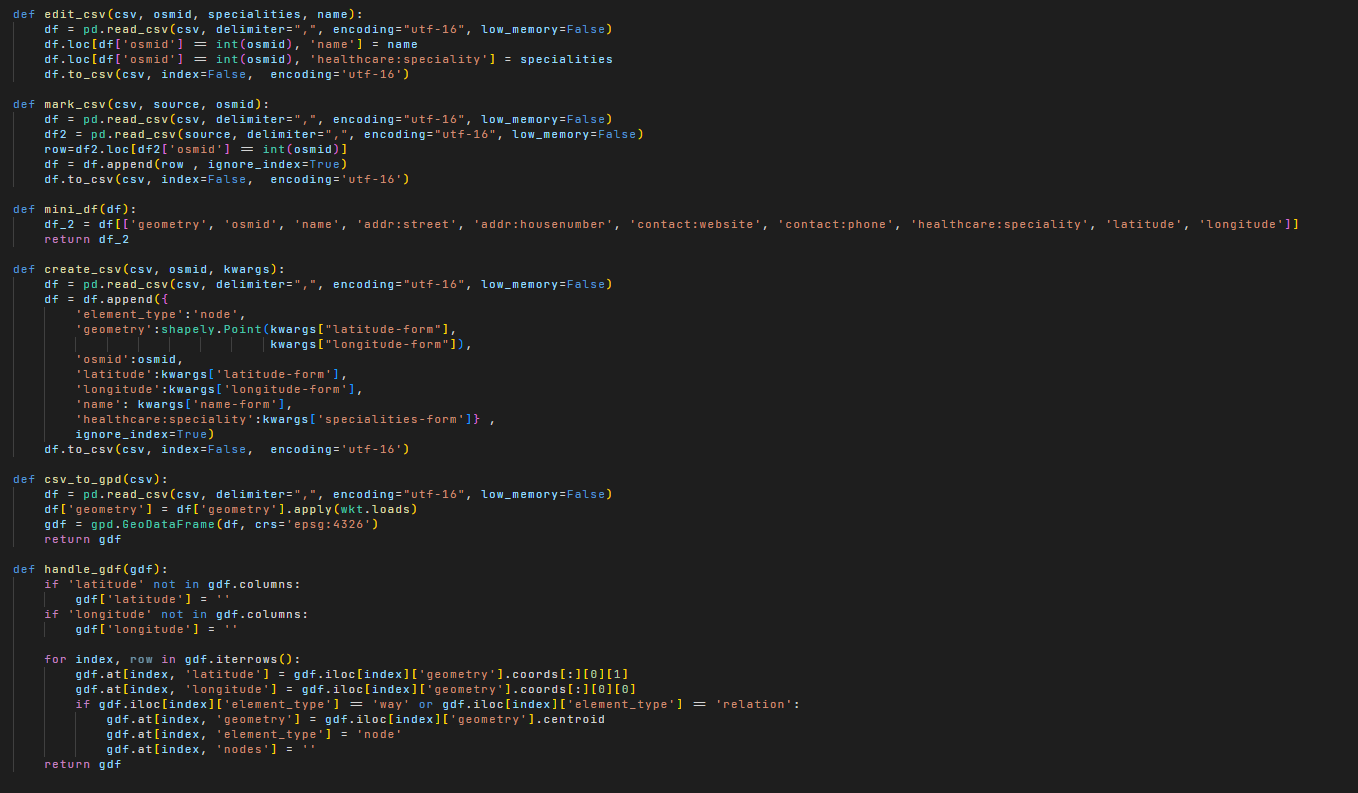


Рисунок 11 – модуль utils

В разделе 2.3.2. были описаны все функции за исключением handle\_gdf. Я не стал включать её в запросы, так как это функция, которая была использована лишь один раз. Она была создана для того, чтобы преобразовать все объекты с типами way и polygon в тип node, а также добавить в каждую строку ширину и долготу.

**3.1.2. Модуль app.py**

На каждое действие (маркировка, редактирование, добавление клиники) был создан роут (mark, post, create соответственно). Их описание нецелесообразно по причине низкой информативности, поэтому будет описана только главная часть модуля. Главной частью является функция main, в которой происходят следующие действия:

1. в переменную content загружаются геоданные всех клиник;
2. в переменную content2 загружаются геоданные всех маркеров;
3. далее данные фильтруются по специальности, если это необходимо;
4. данные переводятся в json и передаются в html шаблон.

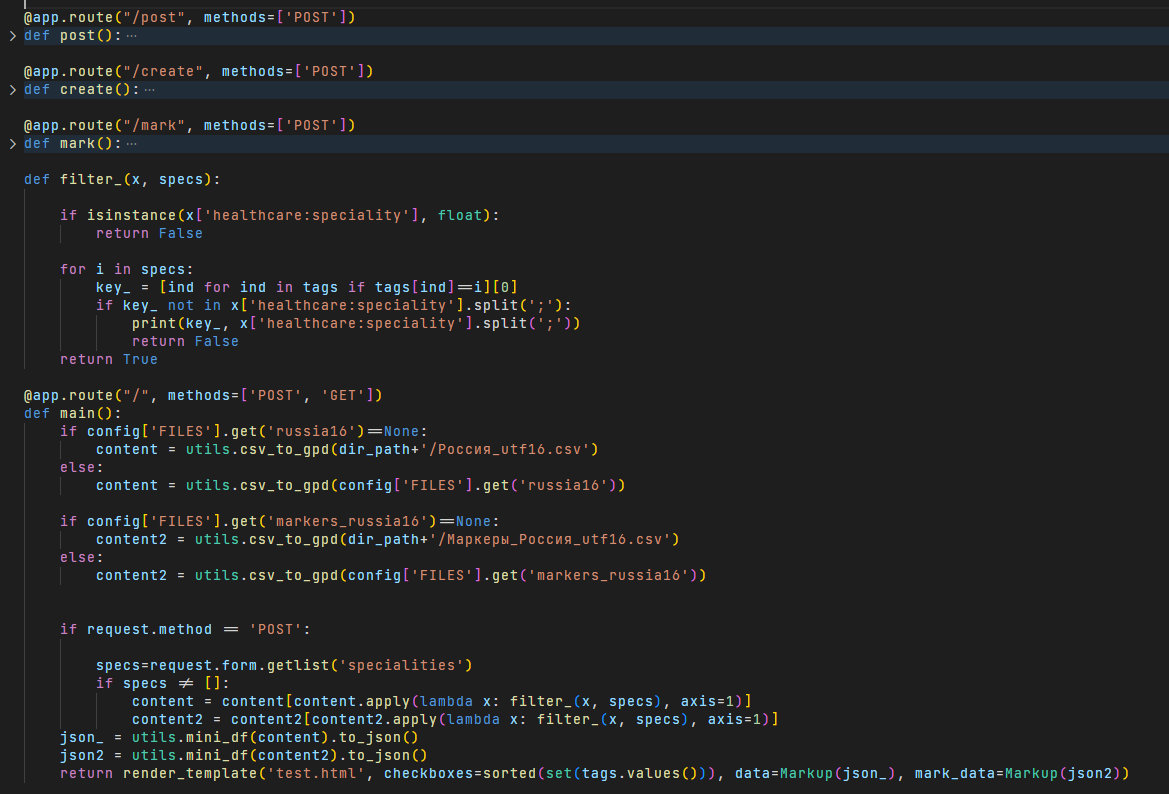


Рисунок 12 – модуль app

**3.1.3. Шаблон test.html**

Шаблон test.html имеет небольшую DOM- структуру, однако длинный JS-код, поэтому будут показаны лишь самые важные его участки, такие как:

1. Создание слоя из переданного JSON объекта
2. Создание самой карты с геоданными
3. Создание формы для редактирования элемента

**3.1.3.1. Создание слоя из переданного JSON объекта**

Как уже было вышеупомянуто, в шаблон test передаются геоданные из App.py. Как мы знаем, шаблонизатор jinja воспринимает переменную, заключенные в две фигурные скобки, как переданную переменную. В данном участке кода сначала создаётся простой рендерер для маркеров, затем в константу передаётся json, после этого он переводится в двоичный вид и после этого из него создаётся слой. Аналогично создаётся и слой для обычных объектов, с той лишь разницей, что в слое для обычных объектов более сложный рендерер.

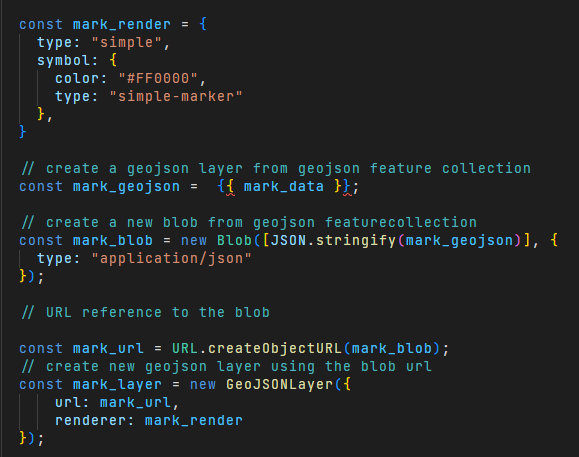


Рисунок 13 – создание слоя

**3.1.3.2. Создание слоя из переданного JSON объекта**

На данном участке кода мы видим, как сначала создаётся карта из двух слоёв, а затем она добавляется в вид. Добавление в вид – непосредственно размещение карты в HTML. Слои layer и mark\_layer – тепловая и маркерная карты клиник соответственно.

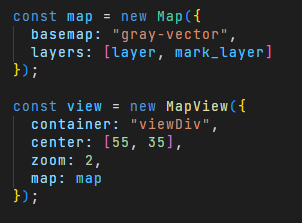


Рисунок 14 – создание карты

**3.1.3.3. Создание формы**

Для того, чтобы иметь возможность редактировать информацию о клиниках, сначала была предпринята попытка изменить под себя встроенное в фреймворк всплывающее окно, однако эта попытка окончилась провалом, по итогу которого было принято решение создать в уведомлении кнопку, которая бы просто создавала форму для изменения клиники. Код приведён ниже.

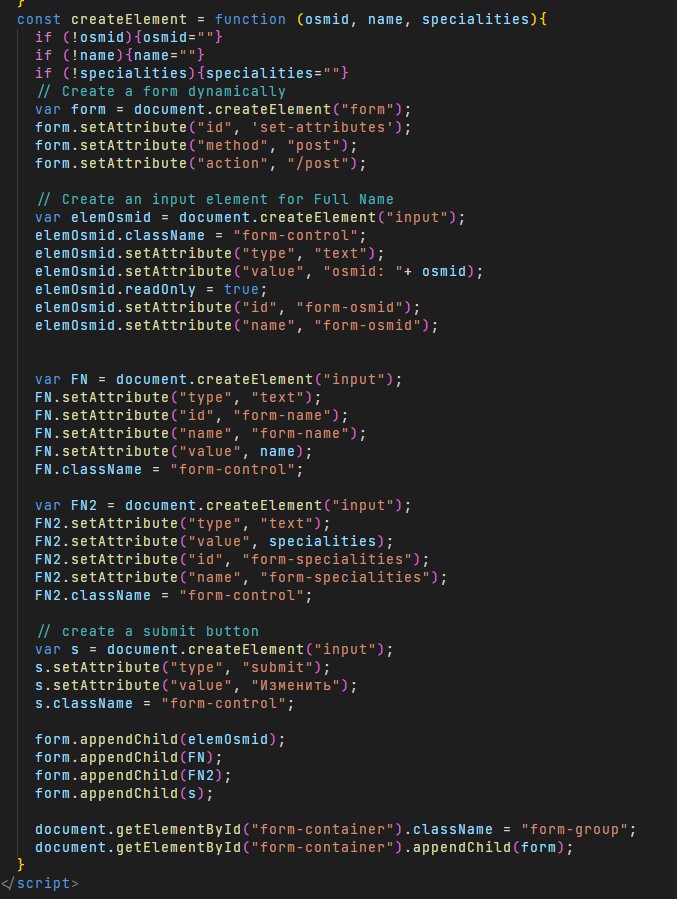


Рисунок 15 – создание формы

**3.2. Финальный вид приложения**

По итогу работы сервис принял следующий вид:

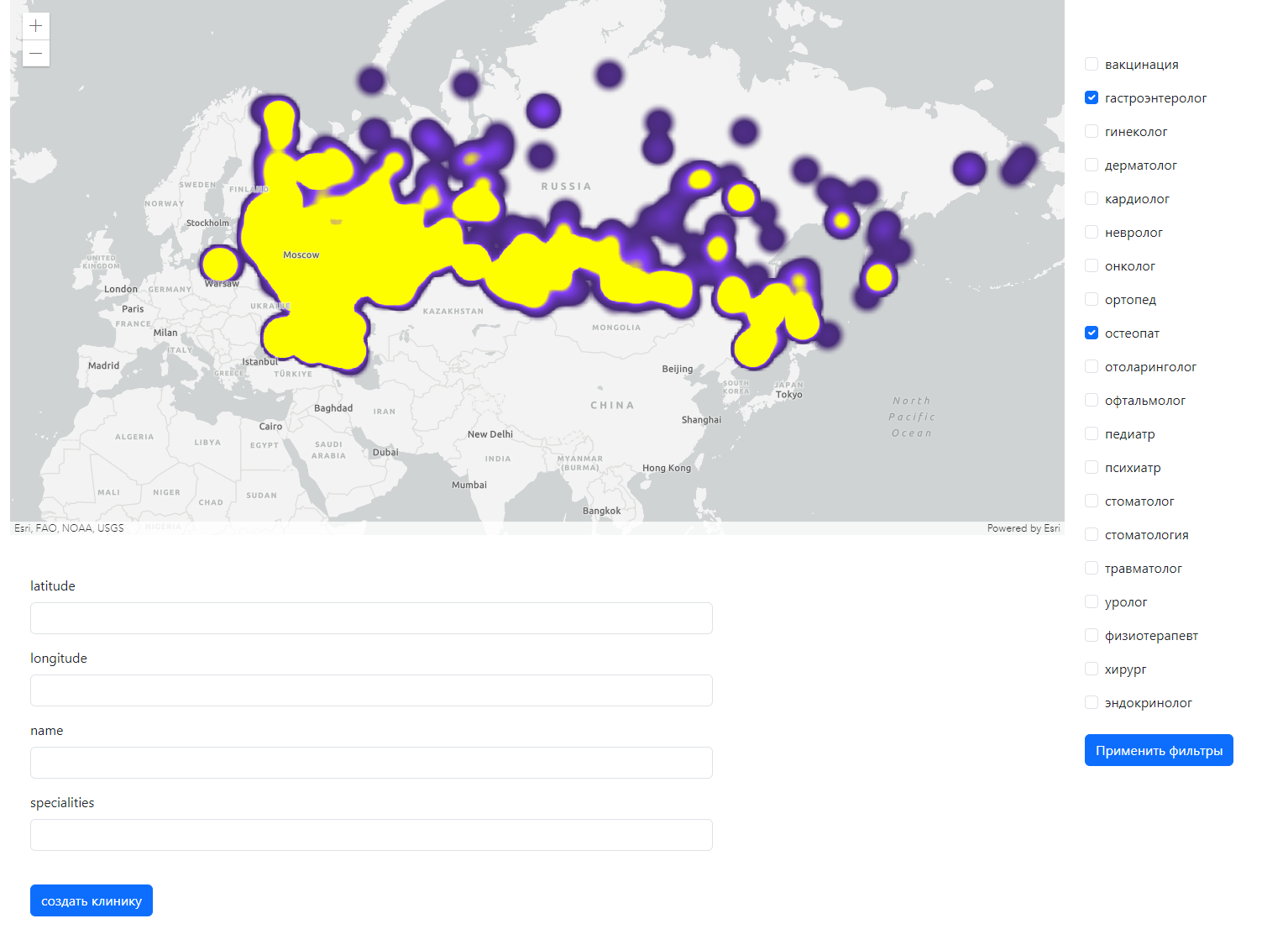


Рисунок 16 – итоговый вид

**Заключение**

В результате курсовой работы были получены навыки работы с геоданными и JS-фреймворками. Также были закреплены навыки работы с Bootstrap.

# Список литературы

1. Официальная документация к фреймворку ArcGis– [Электронный ресурс]. – URL: https://developers.arcgis.com/javascript/latest/api-reference/
2. Официальная документация к библиотеке osmnx– [Электронный ресурс]. – URL: https://osmnx.readthedocs.io/
3. Официальная документация к фреймворку Flask 2.2– [Электронный ресурс]. – URL: https://flask.palletsprojects.com/en/2.2.x/