Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

|  |  |  |  |
| --- | --- | --- | --- |
| Институт: | ИВТИ | Кафедра: | ВМСС |
| Направление подготовки/специальность: | | 09.03.01 Информатика и вычислительная техника | |

**ОТЧЕТ по практике**

|  |  |
| --- | --- |
| **Наименование практики:** | Производственная практика: преддипломная практика |

**СТУДЕНТ**

|  |  |
| --- | --- |
|  | / Палагина С.А. / |
| *(подпись )* | (*Фамилия и инициалы*) |

|  |  |
| --- | --- |
| Группа | А-12-17 |
|  | *(номер учебной группы)* |

**ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ**

|  |
| --- |
|  |
| *(отлично, хорошо, удовлетворительно, неудовлетворительно, зачтено, не зачтено)* |

|  |  |
| --- | --- |
|  | / / |
| *(подпись )* | (*Фамилия и инициалы члена комиссии*) |

|  |  |
| --- | --- |
|  | / / |
| *(подпись )* | (*Фамилия и инициалы члена комиссии*) |

**Москва**

**2021**

Оглавление

[Введение 2](#_Toc73096564)

[1. Постановка задачи 3](#_Toc73096565)

[2. Разработка структуры базы данных 4](#_Toc73096566)

[2.1. Таблица Станций 4](#_Toc73096567)

[2.2. Таблицы Среднемесячных часовых сумм суммарной/прямой/диффузной солнечной радиации на горизонтальную поверхность 5](#_Toc73096568)

[2.3. Таблицы Среднемесячных суточных сумм суммарной/прямой/диффузной солнечной радиации на горизонтальную поверхность 7](#_Toc73096569)

[2.4. Таблицы Среднемесячных месячных сумм суммарной/прямой/диффузной солнечной радиации на горизонтальную поверхность 8](#_Toc73096570)

[2.5. Таблица Альбедо 10](#_Toc73096571)

[2.6. Структура базы данных 11](#_Toc73096572)

[3. Разработка алгоритма интерполяции данных 12](#_Toc73096573)

[3.1. Интерполяция “День сурка” 13](#_Toc73096574)

[3.1.1.Описание принципа итерполяции типа “День сурка” 13](#_Toc73096575)

[3.1.2. Описание алгоритма интерполяции 13](#_Toc73096576)

[3.2. Интерполяция “Линейная” 15](#_Toc73096577)

[3.2.1 Описание принципа интерполяции типа “Линейная” 15](#_Toc73096578)

[3.2.2. Описание алгоритма интерполяции 17](#_Toc73096579)

[4. Экспорт в csv 21](#_Toc73096580)

[4.1. Формат выходных файлов 22](#_Toc73096581)

[4.2. Описание алгоритма экспорта интерполированных данных 23](#_Toc73096582)

[5. Тестирование алгоритма экспорта 26](#_Toc73096583)

[Заключение 31](#_Toc73096584)

[Список использованной литературы 31](#_Toc73096585)

[Приложение. Листинг серверной части программы 31](#_Toc73096586)

**Введение**

Еще в 2004 году на кафедре инстита ГВИЭ была разработана база данных, хранящая среднемноголетние данные о приходе солнечной радиации на горизонтальную поверхность при средних условиях облочности. Данные, содержащиеся в базе, были взяты из «Научно-прикладного справочника по климату СССР». Данные, публикуемые в научно-прикладном справочнике, были рассчитаны и обобщены по принципу максимальной информативности многолетних наблюдений метеорологических станций. Справочник выпускался в 1990-х годах и состоял из шести частей, каждая из которых освещала какой-либо регион бывшего СССР.

Сотрудники института ГВИЭ проделали огромную работу по анализу и структурированию такого огромного количества данных. База данных была создана в формате excel файла. Компьютерная программа Excel сама по себе является мощным аналитическим инструментом, предназначеным для хранения, организации и анализа данных, а так же для решения многих вычислительных задач. Существующая база данных граматно организована, имеет удобный функционал и хорошо структурирована. Но тут возникает вопрос о том, как предоставлять для пользования эту базу данных все желающим. Живя в век информационных технологий, уже не совсем рационально распространять подобного рода информацию на электронных носителях, особенно, если это не один или два человека, а, допустим, целый поток студентов. Намного удобнее создать веб-интерфейс по предоставлению информации, чтобы любой желающий смог перейти по ссылке в интернете и найти все нужные данные без особых трудностей.

Институт ГВИЭ предложил решить вышеупомянутую задачу. С этой целью было разработано веб-приложение для поиска данных о приходе солнечной радиации на горизонтальную поверхность, с последующей их обработкой и возможностью скачивания искомой информации. Данное приложение находится в свободном доступе, то есть каждый желающий сможет просматривать данные. Не было смысла закрывать информацию, которая и так была общедоступна. В рамках данного приложения не было предусматрено предоставление пользователям разных привелегий доступа, поскольку никакие новые данные в базу не добавляться, а существующие не будут подвергаться изменению. Такие послабления также исходят и того факта, что исходные данные – это среднемноголетние наблюдения, сформированные уже достаточно давно, то есть они являются, своего рода, базовыми.

**Постановка задачи**

Базы данных, с которой предстояло работать, содержит среднемноголетние данные о приходе солнечной радиации на горизонтальную поверхность для различных актинометрических станций, которые располагаются в регионах нашей страны и на территории бывшей СССР. Для удобства работы все эти данные были перенесены в реляционную базу данных MySQL.

Основные требования к базе данных:

* данные должны быть структурированы;
* должна быть исключена избыточность информации, ее дублирование и противоречивость;
* база данных должна быть сформирована таким образом, чтобы запросы для обращения к ней были как можно проще;
* база данных должна быть масштабируема, то есть она должна быть построена таким образом, чтобы было возможно ее дальнейшее расширение и изменение.

Веб-приложение – это механизм, для доступа к вышеупомянутой базе данных. Из этого следует, что функционал самого приложения будет не слишком сложным, тем более отстуствует возможность добавления, удаления и обновления базы данных. Следовательно, задача сводится к разработке удобного интерфейса для поиска данных. Помимо этого данные еще необходимо в некоторых случаях интерполировать, а полученный результат поиска скачивать.

Требования, предъявляемые к разрабатываемомму приложению:

* удобство использования – пользовательский интерфейс должен быть интуитивно понятным;
* наглядность – запрашиваемая информация должна корректно и понятно отображаться для просмотра;
* настраиваемость – пользователь должен иметь возможность задать подходящие ему параметры для интерполяции, такие как: тип интерполяции, шаг дискретизации и период, за который необходимо получить данные;
* возможность скачивания – пользователю должна быть предаставлена возможность экспортировать полученные данные.

Формулировка поставленной задачи представлена в следующем виде: необходимо разработать веб-интерфейс для удобного поиска информации по базе данных, содержащей информацию по актинометрическим станциям, с последующим интерполированием полученных данных, с возможностью задания параметров для интерполяции, а также с наличием функционала для экспорта готовых данных в формате csv.

Веб-приложение состоит из трех основных блоков: поиск, интерполяция и экспорт.

1. Поиск - предоставление возможности поиска актинометрической станции по нескольким основным параметрам: по названию станции, по номеру станции и по географическим координатам станции.

2. Интерполяция – предоставление возможности интерполировать среднемесячные значения прихода солнечной радиации на горизонтальную поверхность в зависимости от заданных параметров: тип интерполяции, шаг дискретизации и период, за который необходимо получить данные.

3. Экспорт – предоставление возможности экспортировать либо исходные данные, либо интерполированные в формате csv, с заданной структурой файла.

**Средства разработки**

Первое, что необходимо сделать при создании веб-приложения, - это проанализировать все доступные средства, для быстрой и качественной разработки. Этими средствами являются язык программирования, фреймворк и СУБД**.**

**Выбор языка программирования**

Сначала предстоит выбрать язык программирования, который поможет создать многофункциональное и удобное веб-приложение. Сравним два наиболее популярных языка, используемых в веб-разработке: Python и PHP.

Несмотря на то, что PHP и Python пользуются большим спросом, каждый из них служит разным целям. В то время как PHP изначально разрабатывался как веб-язык, Python создавался как язык программирования общего назначения. И PHP, и Python являются динамически типизированными и объектно-ориентированными языками, которые совместимы с несколькими операционными системами, поэтому в некотором смысле они похожи [13].

Сравним два языка.

1. Сложность

Python и PHP отличаются в простоте использования.

PHP изначально разрабатывался как инструмент для создания динамических веб-сайтов и веб-приложений.  Разработка PHP может быть слишком сложной из-за его жесткого синтаксиса. Например, названия функций могут быть похожи, но выполнять совсем разные операции. Если в коде будет ошибка, то язык позволит использовать ее. Когда недостаток будет очевидным, ошибку будет очень сложно найти [13,14].

Напротив, Python довольно прост для понимания благодаря его высокой читабельности. Кроме того, в Python написанный код можно легко интерпретировать и читать, что упрощает процесс отладки, поскольку в простом синтаксисе легче найти ошибки и уязвимости. Python помогает упростить сложное программирование. Он внутренне обрабатывает адреса памяти, уборщиков мусора [13,14].

2. Безопасность

Python считается одним из самых безопасных языков программирования. Python имеет несколько функций безопасности, которые можно использовать для создания сложных приложений с точными целями и функциями. Например, его фреймворк Django имеет встроенные функции безопасности, которые позволяют разработчикам эффективно бороться с угрозами.

В отличие от Python, большое количество приложений PHP может иметь проблемы с безопасностью из-за старых методов кодирования и плохого кода [13,14].

3. Популярность

И Python, и PHP имеют отличную поддержку сообщества. PHP присутствует на рынке с 1995 года и сумел сформировать огромное сообщество разработчиков, готовых оказать поддержку.

Python был выпущен еще раньше, в 1991 году. Как и его конкурент, Python имеет большое сообщество разработчиков, которые непрерывно разрабатывают веб-приложения, и, следовательно, поддержка сообщества является выдающейся [13,14].

Поэтому, учитывая популярность языков, сложностей в поиске ответов на возникающие вопросы возникнуть не должно в обоих случаях.

Сравнивая и другие параметры языков, такие как: наличие фреймворков, подключение к базе данных, масштабируемость, было решено остановиться на языке программирования Python. Данный язык очень прост в освоении, имеет высокое качество кода, имеет внушительное количество различных библиотек, хоть и немного проигрывает PHP производительности и не поддерживает подключение к базе данных так широко, как PHP, однако, эти недостатки в рамках разработки нашего веб-приложения не вызовут явных неудобств.

**Выбор фреймворка**

Фреймворк – это рабочая среда, которая помогает разработчику быстро и качественно создавать программный продукт, не отвлекаясь на мелочи.  
Они оснащены полезными функциями, такими как шаблоны, механизмы управления сеансами и библиотеки доступа к базам данных.

Django и Flask – два очень широко используемых фреймворка для Python. Оба фреймворка имеют открытый исходный код.

Django был создан для упрощения процесса разработки сайта. Он предлагает веб-разработчикам набор компонентов, которые увеличивают скорость разработки и облегчают весь процесс.

Flask – это небольшой веб-фреймворк, который не требует использования определенных библиотек или инструментов. Хоть в нем и нет компонентов, интегрирующих сторонние библиотеки, Flask поддерживает расширения, позволяющие легко подключить недостающие функции [15, 16].

Посмотрим чем они отличаются.

1. ORM

Django предоставляет свой собственный Django ORM (объектно-реляционное отображение) и использует модели данных, в то время как Flask вообще не имеет моделей данных и в принципе поставляется без ORM, следовательно, придется подключать стороннюю ORM. Легче использовать уже готовое решение.

2. Встроенные пакеты

Flask очень минималистичный, поэтому нам необходимо использовать внешние библиотеки в зависимости от поставленной задачи. Это делает Flask гибким и расширяемым.

Django имеет огромное количество встроенных пакетов, следовательно, поиск пакета для сборки и запуска приложения пройдет с меньшими усилиями.

3. Область использование

Django был разработан для быстрой разработки сложных веб-приложений. Он предоставляет необходимые инструменты для реализации масштабируемой и поддерживаемой функциональности. С другой стороны, простота Flask позволяет быстрее создавать небольшие приложения.

Также стоит учесть, что Django автоматически создает строго структурированную систему директорий при создании приложения. Если же мы используем Flask, то нам необходимо самим продумывать структуру приложения, что может вызвать дополнительные сложности.

Основное различие между Django и Flask в том, что Django предоставляет полнофункциональную среду Model-View-Controller («Модель–Представление–Управление»). В Django паттерн MVC чаще обозначают как MTV («Модель– Шаблон–Представление»).

В концепции MTV:

* «Model» — модель, уровень доступа к данным. Сюда относится всё, что связано с данными — как получить к ним доступ, как проверить их, какое у них поведение и отношение друг с другом.
* «Template» — шаблоны. Этот уровень содержит в себе всё, что связано с отображением: как именно что-то должно быть отображено на веб-странице или любом другом документе.
* «View» — представление. Этот уровень связан со всей логикой, связывающей модель и необходимые шаблоны. Вы можете себе представлять это уровень как мост между моделями и шаблонами.

Происходит разделение данных приложения, пользовательского интерфейса и логики на три отдельных компонента, поэтому изменение отдельного компонента может осуществляться независимо [17].

В результате анализа всех параметров для разработки веб-приложения был выбран фреймворк Django.

**Выбор СУБД**

По умолчанию в настройках указано использование SQLite. SQLite уже включен в Python, следовательно, не нужно дополнительно что-то устанавливать [3].

SQLite — компактная встраиваемая реляционная база данных. Имеет открытый исходный код библиотеки. SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. К недостаткам относятся Отсутствие пользовательского управления, то есть отсутствие возможности управлять связями в таблицах в соответствии с привилегиями и **невозможность дополнительной настройки, то есть** SQLite нельзя сделать более производительной [18].

Поэтому в качестве СУБД была выбрана реляционная база данных MySQL.

MySQL - это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями.

К преимуществам данной базы данных относятся:

* Простота: MySQL легко устанавливается. Существует много сторонних инструментов, включая визуальные, облегчающих начало работы с БД.
* Много функций: MySQL поддерживает большую часть функционала SQL.
* Безопасность: в MySQL встроено много функций безопасности.
* Мощность и масштабируемость: MySQL может работать с действительно большими объёмами данных, и неплохо походит для масштабируемых приложений [18].

**Разработка структуры базы данных**

В данном разделе описан процесс разработки реляционной базы данных в СУБД MySQL. В процессе создания была проанализирована предоставленная институтом база данных, содержащая в себе среднемноголетние данные по активным станциям, такие как:

1. Приход солнечной радиации (суммарной, прямой, диффузной) на горизонтальную поверхность при средних условиях облачности:

1) Среднемесячные часовые суммы солнечной радиации

2) Среднемесячные суточные суммы солнечной радиации

3) Месячные суммы солнечной радиации

2. Среднемесячные коэффициенты отражательной способности поверхности – альбедо.

Всего общее количество наблюдаемых метрологических станций – 166.

На основе полученной информации была проработана структура таблиц, а также взаимосвязи между ними. Основной целью было устранить дублирование и избыточность информации. Также важно было избежать противоречивости в хранимых данных.

Для достижения данных целей было необходимо провести нормализацию всех таблиц базы данных, то есть привести их к нормальным формам.

Нормализация – это процесс последовательной пошаговой модификации таблиц базы данных, имеющий целью сокращение избыточности и несогласованности. Процесс нормализации устроен таким образом, что по завершении его очередного шага нормализуемая база данных переходит в определенную форму, называемую нормальной и связанную с этим шагом. Так, реляционная модель определяет три нормальных формы.

* первая нормальная форма – каждой столбец в строке должен быть атомарным, каждая строка имеет одинаковое количество столбцов, все строки в таблице должны быть разными;
* вторая нормальная форма – таблица должна удовлетворять требованиям первой нормальной формы и все ее поля, не входящие в первичный ключ, должны зависеть от полного первичного ключа;
* третья нормальная форма – таблица должна удовлетворять требованиям второй нормальной формы и все ее столбцы, не входящие в первичный ключ, должны зависеть от первичного ключа и не зависеть друг от друга.

При этом каждая последующая нормальная форма сильнее предыдущей в том смысле, что удовлетворяет критериям предыдущей нормальной формы [9].

Исходя из данных требований, были разработаны таблицы базы данных, которые рассмотрены в следующем разделе.

**Разработка таблиц**

**Таблица Станции**

Первой и основополагающей таблицей является таблица, содержащая необходимые данные по каждой из 166 станций, которые могут потребоваться для её описания.

Естественно, таблица должна иметь свой первичный ключ, а именно уникальный идентификатор id, который будет внешним ключом для остальных таблиц. При помощи этого поля мы связываем все наши таблицы. Столбцу *id* соответствует номер конкретной станции.

Также для полного описания станции мы должны хранить в данной таблице название этой станции – «Name\_Station», регион ее расположения – «Region», а также географические координаты ее расположения: широта – «Latitude» и долгота – «Longitude».

В Табл. 1 представлена структура таблицы «Станции».

**Табл. 1**

**Таблица «Станции»**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип данных | Описание переменной |
| id | int | Идентификатор станции, является уникальным (первичный ключ) |
| Name\_Station | Varchar(50) | Название станции |
| Region | Varchar(50) | Регион расположения станции |
| Latitude | Decimal(5,2) | Широта |
| Longitude | Decimal(5,2) | Долгота |

**Таблицы Среднемесячных часовых сумм суммарной, прямой и диффузной солнечной радиации на горизонтальную поверхность**

Проанализируем структуру фрагмента таблицы Среднемесячных часовых сумм суммарной, прямой и диффузной солнечной радиации из предоставленной базы данных для конкретной станции, которая показана на Рис. 1.

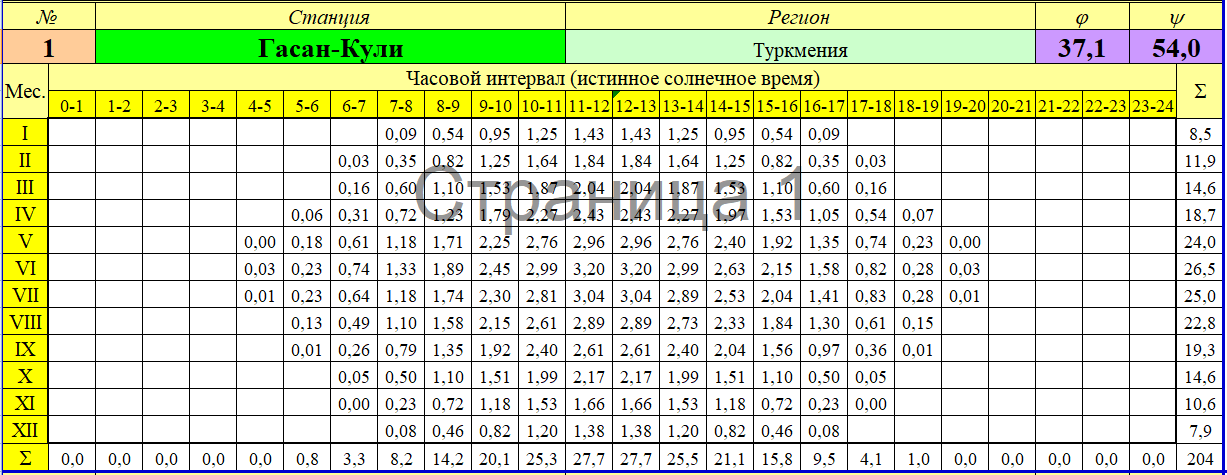


Рис. 1 Структура таблицы часовых сумм

Как было замечено, структура хранения данных здесь достаточно сложная. Можно наблюдать повторяющиеся данные, которые относятся к описанию станции. Это приводит к избыточности, так как вся эта информация уже хранятся в таблице Stations. Естественно, в будущей таблице необходимо указать только id станции, чтобы идентифицировать наши данные для конкретной станции.

Общая структура исходной таблицы такова, что по строкам идут месяцы в году, а по столбцам часы в сутках. Для дальнейшего удобства работы с данными нам необходимо транспонировать данную таблицу, то есть поменять местами строки и столбцы.

Исходя из специфики хранения исходных данных, была разработана таблица с составным первичным ключом.

Первой частью ключа будет внешний ключ от таблицы Станции – «id», который является уникальным идентификатор станции. Благодаря этой части составного первичного ключа явно указываем, для какой конкретно станции храним данные. Второй частью ключа будет id часа в сутках - «*ID-Hour*». Полученное сочетание id станции и id часа в сутках даст уникальную комбинацию для хранения значений по месяцам в базе данных MySQL.

Остальные столбцы таблицы соответствуют каждому месяцу в году.

Рассмотрим структуру данных таблиц на примере среднемесячных часовых сумм суммарной солнечной радиации, которая предствалена в Табл. 2.

**Табл. 2**

**Таблица « Среднемесячных часовых сумм суммарной солнечной радиации »**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип данных | Описание переменной |
| ID\_Station | int | Идентификатор станции (внешний ключ является первичным ключом) |
| ID\_Hour | int | Номер часа в сутках |
| Month\_1 | Decimal(5,2) | Здесь будут храниться суммы по всем часам для конкретного месяца |
| Month\_2 | Decimal(5,2) |
| Month\_3 | Decimal(5,2) |
| Month\_4 | Decimal(5,2) |
| Month\_5 | Decimal(5,2) |
| Month\_6 | Decimal(5,2) |
| Month\_7 | Decimal(5,2) |
| Month\_8 | Decimal(5,2) |
| Month\_9 | Decimal(5,2) |
| Month\_10 | Decimal(5,2) |
| Month\_11 | Decimal(5,2) |
| Month\_12 | Decimal(5,2) |

**Таблицы Среднемесячных суточных сумм суммарной, прямой и диффузной солнечной радиации на горизонтальную поверхность**

Проанализируем структуру фрагмента таблицы Среднемесячных суточных сумм суммарной/прямой/диффузной солнечной радиации из предоставленной базы данных для конкретной станции, которая показана на Рис. 2.

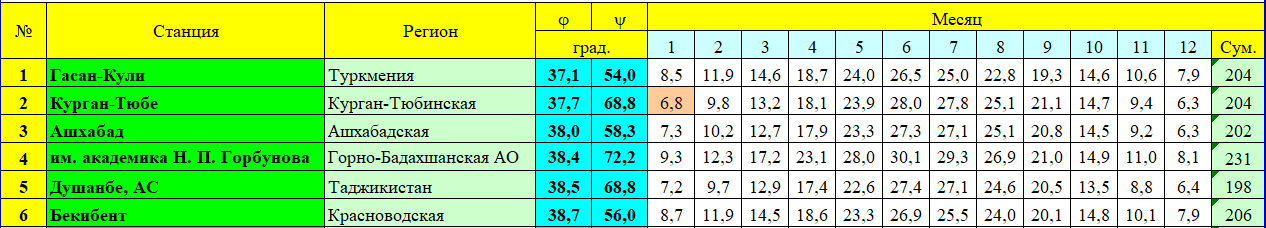


Рис. 2 Структура таблицы суточных сумм

Данные здесь так же повторяются, а именно: номер станции, название, регион, широта и долгота. Было не правильно переносить в базу данных таблицу в таком виде, поскольку мы получим огромную избыточность. Поэтому была разработана таблица, в которой первичным ключом является внешний ключ от таблицы «Stations» - «id», который является уникальным индентификатором станции. Остальные столбцы таблицы соответствуют месяцу в году.

Рассмотрим структуру данных таблиц на примере среднемесячных суточных сумм суммарной солнечной радиации, которая предствалена в Табл. 3.

**Табл. 3**

**Таблица «Среднемесячных суточных сумм суммарной солнечной радиации»**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип данных | Описание переменной |
| ID\_Station | int | Идентификатор станции (внешний ключ является первичным ключом) |
| Month\_1 | Decimal(5,2) | Хранит данные для данной станции за январь |
| Month\_2 | Decimal(5,2) | Хранит данные для данной станции за февраль |
| Month\_3 | Decimal(5,2) | Хранит данные для данной станции за март |
| Month\_4 | Decimal(5,2) | Хранит данные для данной станции за апрель |
| Month\_5 | Decimal(5,2) | Хранит данные для данной станции за май |
| Month\_6 | Decimal(5,2) | Хранит данные для данной станции за июнь |
| Month\_7 | Decimal(5,2) | Хранит данные для данной станции за июль |
| Month\_8 | Decimal(5,2) | Хранит данные для данной станции за август |
| Month\_9 | Decimal(5,2) | Хранит данные для данной станции за сентябрь |
| Month\_10 | Decimal(5,2) | Хранит данные для данной станции за октябрь |
| Month\_11 | Decimal(5,2) | Хранит данные для данной станции за ноябрь |
| Month\_12 | Decimal(5,2) | Хранит данные для данной станции за декабрь |

**Таблицы Среднемесячных месячных сумм суммарной, прямой и диффузной солнечной радиации на горизонтальную поверхность**

Проанализируем структуру фрагмента таблицы Среднемесячных месячных сумм суммарной, прямой и диффузной солнечной радиации из предоставленной базы данных для конкретной станции, которая показана на Рис. 3. Поскольку она аналогична таблице, представленной на Рис.2, то и структура формируемой таблицы будет похожей.

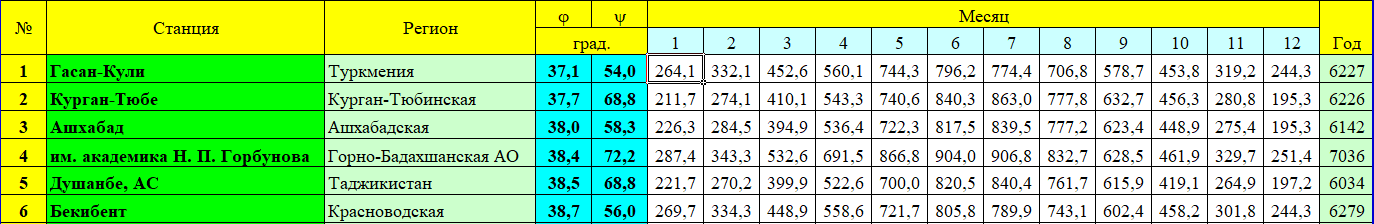
****

Рис. 3 Структура таблицы месячных сумм

Была разработана таблица, в которой первичным ключом будет внешний ключ от таблицы «Stations», а именно уникальный идентификатор станции. Остальные столбцы таблицы соответствуют месяцу в году.

Рассмотрим структуру данных таблиц на примере среднемесячных часовых сумм суммарной солнечной радиации, которая предствалена в Табл. 4.

**Табл. 4**

**Таблица «Среднемесячных месячных сумм суммарной солнечной радиации»**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип данных | Описание переменной |
| ID\_Station | int | Идентификатор станции (внешний ключ является первичным ключом) |
| Month\_1 | Decimal(5,2) | Хранит данные для данной станции за январь |
| Month\_2 | Decimal(5,2) | Хранит данные для данной станции за февраль |
| Month\_3 | Decimal(5,2) | Хранит данные для данной станции за март |
| Month\_4 | Decimal(5,2) | Хранит данные для данной станции за апрель |
| Month\_5 | Decimal(5,2) | Хранит данные для данной станции за май |
| Month\_6 | Decimal(5,2) | Хранит данные для данной станции за июнь |
| Month\_7 | Decimal(5,2) | Хранит данные для данной станции за июль |
| Month\_8 | Decimal(5,2) | Хранит данные для данной станции за август |
| Month\_9 | Decimal(5,2) | Хранит данные для данной станции за сентябрь |
| Month\_10 | Decimal(5,2) | Хранит данные для данной станции за октябрь |
| Month\_11 | Decimal(5,2) | Хранит данные для данной станции за ноябрь |
| Month\_12 | Decimal(5,2) | Хранит данные для данной станции за декабрь |

**Таблица Альбедо**

Проанализируем структуру фрагмента таблицы из предоставленной базы данных для конкретной станции, которая показана на Рис. 4. Таблица содержит среднемесячные коэффициенты отражающей способности поверхности – альбедо (о.е.) для всех станций.

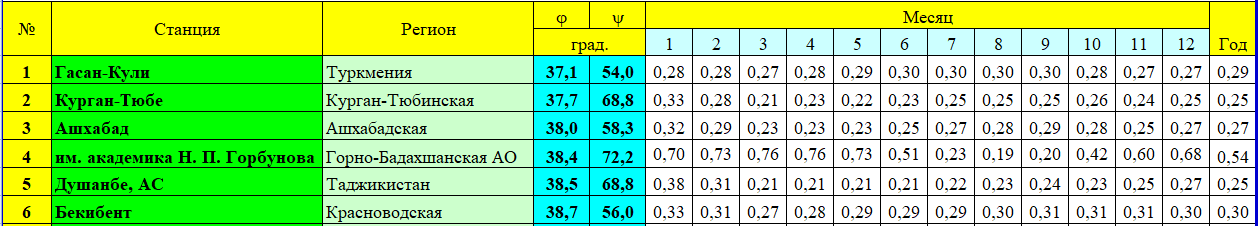


Рис. 4 Структура таблицы Альбедо

В разработынной таблице, первичным ключом будет внешний ключ от таблицы «Stations», то есть уникальный идентификатор станции. Остальные столбцы таблицы соответствуют месяцу в году.

В Табл. 5 представлена структура таблицы Альбедо.

**Табл. 5**

**Таблица «Альбедо»**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип данных | Описание переменной |
| ID | int | Идентификатор станции (внешний ключ является первичным ключом) |
| Month\_1 | Decimal(5,2) | Хранит данные для данной станции за январь |
| Month\_2 | Decimal(5,2) | Хранит данные для данной станции за февраль |
| Month\_3 | Decimal(5,2) | Хранит данные для данной станции за март |
| Month\_4 | Decimal(5,2) | Хранит данные для данной станции за апрель |
| Month\_5 | Decimal(5,2) | Хранит данные для данной станции за май |
| Month\_6 | Decimal(5,2) | Хранит данные для данной станции за июнь |
| Month\_7 | Decimal(5,2) | Хранит данные для данной станции за июль |
| Month\_8 | Decimal(5,2) | Хранит данные для данной станции за август |
| Month\_9 | Decimal(5,2) | Хранит данные для данной станции за сентябрь |
| Month\_10 | Decimal(5,2) | Хранит данные для данной станции за октябрь |
| Month\_11 | Decimal(5,2) | Хранит данные для данной станции за ноябрь |
| Month\_12 | Decimal(5,2) | Хранит данные для данной станции за декабрь |

**2.6. Соединение веб-приложения с базой данных**

Серверное приложение разрабатывается с помощью фреймворка Django 3.1.6 на языке Python 3.9. Поскольку Django по умолчанию работает с СУБД SQLite, то для работы с другими СУБД необходимо прописать настройках проекта (файл «settings.py»), с какой базой данных необходимо работать и как к ней подключаться (наименование базы данных, логин, пароль, хост, порт). После соединения проекта Django с базой данных можно приступить к созданию моделей в проекте и с помощью миграций создавать или изменять реальные таблицы базы данных.

Модели в Django описывают структуру используемых данных. Используемые в программе данные хранятся в базах данных, и с помощью моделей как раз осуществляется взаимодействие с базой данных. Как правило, каждая модель отображается в одну таблицу базы данных [4].

При создании приложения по умолчанию в его каталог добавляется файл «models.py», который применяется для определения моделей. Модель представляет класс, унаследованный от django.db.models.Model [4].

В файле «models.py» определили модели, поля которых аналогичны столбцам таблиц, которые были разработаны ранее. Далее необходимо осуществить миграции, то есть преобразовать базу данных в соответствии с определением моделей. Миграции осуществляются при помощи двух команд: makemigrations и migrate. Первая команда отвечает за создание миграции, а вторая соответственно за само выполнение миграции. После выполнения миграций в базе данных появится ряд таблиц, которые будут иметь название по имени приложения и модели. Помимо этого появятся также еще и служебные таблицы.

**Структура базы данных**

В базе данных все таблицы связаны только с основной таблицей «mainapp\_stations». Они не связаны между собой, потому что таблицы не содержат взаимосвязанных данных.

Таблицы, содержащие данные о суточных и месячных суммах суммарной, прямой и диффузной солнечной радиации, а также таблица, содержащая среднемесячные коэффициенты отражающей способности поверхности – альбедо, имеют связь с основной таблицей «один к одному», поскольку каждой конкретной станции соответствует одна строка в таблице.

С таблицами, содержащими данные о часовых суммах суммарной, прямой и диффузной солнечной радиации, немного другой случай. Одной станции соответствуют сразу несколько строк в таблице. Данная ситуация складывается из-за первичного ключа, который состоит из id станции и id часа, то есть мы получаем 24 строки для каждой станции для каждого часа в сутках. Поэтому здесь будет связь «один ко многим». Структура полученной базы данных, с указанием связей, представлена на Рис. 5. В структуре не показаны служебные таблицы, поскольку в проекте они не используются.

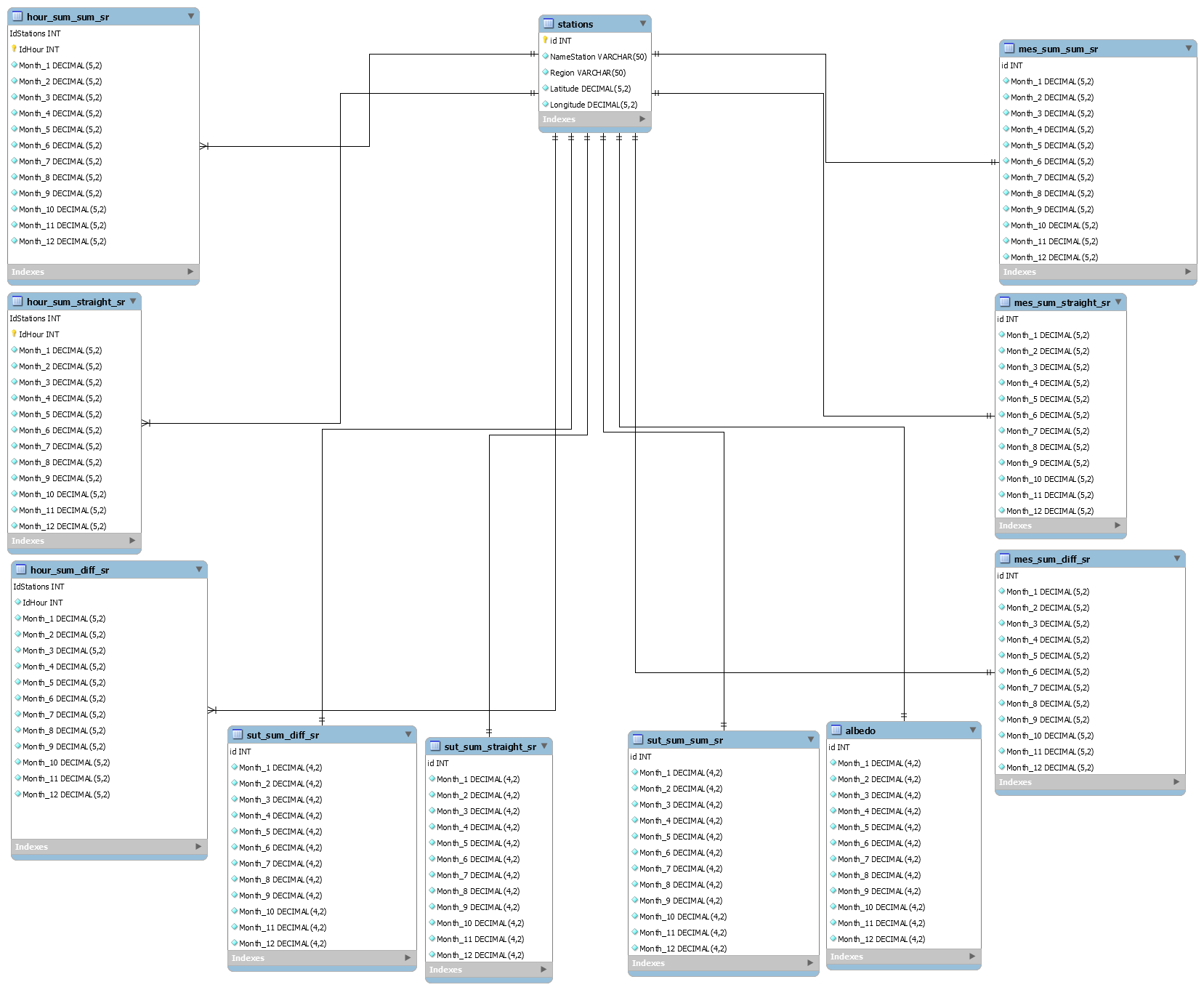


Рис. 5 Структура базы данных MySQL

**3. Архитектура разрабатываемого web-приложения**

Веб-приложение разрабатывалось по архитектуре «клиент-сервер». Из названия понятно, что в данной концепции участвуют две стороны: клиент и сервер.

Клиент – локальный компьютер на стороне виртуального пользователя, который выполняет отправку запроса к серверу для возможности предоставления данных или выполнения определенной группы системных действий.

Сервер – очень мощный компьютер или специальное системное оборудование, которое предназначается для разрешения определенного круга задач по процессу выполнения программных кодов. Он выполняет работы сервисного обслуживания по клиентским запросам, предоставляет пользователям доступ к определенным системным ресурсам, сохраняет данные или БД [11].

Обычно клиент нужен для обеспечения красивого интерфейса, который был бы удобен для конечного пользователя, а сервер для проведения основных вычислений и хранения информации.

Также стоит заметить, что в основе взаимодействия клиент-сервер лежит принцип того, что такое взаимодействие начинает клиент, сервер лишь отвечает клиенту и сообщает о том, может ли он предоставить услугу клиенту и если может, то на каких условиях. Веб-сайты являются частным случаем такой архитектуры, когда общение между клиентом и сервером происходит через сеть Интернет по протоколу HTTP. Роль клиентской части выполняет браузер, который реализует пользовательский интерфейс, формирует запросы к серверу и обрабатывает ответы от него [8].

Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP  (протокол передачи гипертекста).

Данная концепция взаимодействия была разработана в первую очередь для того, чтобы разделить нагрузку между участниками процесса обмена информацией, а также для того, чтобы разделить программный код поставщика и заказчика [10].

**Структура разрабатываемого web-приложения**

Одним из преимуществ фреймворка Django являетя то, что проект всегда имеет конкретный каркас, то есть имеется некоторый набор папок и файлов, которые содержат в себе все основные составляющие проекта. Рассмотрим ниже файловую структуру разработанного веб-приложения.

MainProject/

MainApp /

migrations/

static/

templates/

\_\_init\_\_.py

admin.py

apps.py

forms.py

models.py

tests.py

urls.py

view.py

MainProject /

\_\_init\_\_.py

asgi.py

setting.py

urls.py

wsgi.py

venv/

manage.py

Рассмотрим назначение основных компонент проекта ниже.

MainProject/ – внешний корневой каталог, контейнер для нашего проекта.

MainProject/ – папка проекта.

\_\_init\_\_.py – файл , который сообщает Python, что этот каталог должен рассматриваться как пакет Python’а.

asgi.py - точка входа для ASGI-совместимых веб-серверов для обслуживания проекта.

setting.py – файл содержит в себе все настройки проекта. Здесь регистрируем приложения, задаём размещение статичных файлов, настройки базы данных и так далее.

urls.py – файл, который задаёт ассоциации URL адресов с представлениями. Несмотря на то, что этот файл может содержать *все* настройки url, обычно его делят на части, по одной на приложение

wsgi.py – файл, использующийся для налаживания связи между вашим Django приложением и веб-сервером.

MyApp/ - папка приложения.

migrations/ - папка используется, чтобы хранить «миграции» — файлы, которые позволяют вам автоматически обновлять базу данных по мере изменения моделей.

static/ – папка, содержащая в себе статические файлы, необходимые для работы веб-приложения.

\_\_init\_\_.py – файл указывает, что приложение является пакетом Python.

admin.py – файл, хранящий настройки интерфейса администратора приложения.

apps.py- файл определения информации о приложении. В нем создается класс AppConfig, который используется для определения метаданных, таких как имя приложения.

tests.py – файл используется для проверки файлов кода.

forms.py – файл, используемый для описания форм приложения.

models.py – файл, используемый для определения моделей приложения.

MyApp/ urls.py – файл, который задает ассоциации URL адресов с представлениями в приложении.

view.py – файл, в котором содержатся все выполняемые в приложении представления.

template/ - папка, включающая в себя шаблоны HTML-страниц.

venv/ - виртуальное окружение: все подключаемые внешние модули и библиотеки.

manage.py – скрипт, используемый для взаимодействием с проектом, при его помощи можно создаются новые приложения, работать с базами данных и запускать отладочный сервер.

Рассмотрим функции, используемые в веб-приложении, которые являются содержимым файла «views.py»,.

**Табл.**

**Описание функций в файле «views.py»**

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Входные параметры** | **Назначение** | **Отображаемый шаблон** |
| index | request | Отображение шаблона главной страницы | Index.html |
| ShowStations | request | Отображение шаблона со списком всех станций | ShowStations.html |
| FindName | request | Отображение шаблона для поиска станции по названию | FindName.html |
| FindNumber | request | Отображение шаблона для поиска станции по номеру | FindNumber.html |
| FindCoord | request | Отображение шаблона для поиска станции по географическим координатам | FindCoord.html |
| SearchResult | request | Формирование шаблона с результатами поска | SearchResult.html |
| Export\_csv | request | Экспорт выбранных данных в формате csv | Export\_csv.html |
| Show | request | Отображение шаблона с тремя ближайшими станциями для выбора | Show.html |
| Export\_page | request | Отображение шаблона для задания параметров необходимых для интерполирования | Export\_page.html |
| Export\_csv\_intr | request | Экспорт интерполированных данных в формате csv | - |

Параметр «request» - это объект HttpRequest, содержащий данные о запросе.

**Разработка алгоритмов**

**4. Разработка алгоритмов поиска**

В разрабатываемом приложении для удобства использования необходимо реализовать несколько видов поиска:

* Поиск по названию
* Поиск по номеру станции
* Поиск по географическим координатам

**Поиск по названию**

Изначально необходимо считать введенные данные из формы. Форма состоит всего из одного поля, в которое пользователю необходимо ввести название интересующей станции. Для дальнейшего поиска необходимо передать данные на сервер при помощи метода GET. Метод используется по причине того, что нам необходимо осуществить простой поиск по базе данных без внесения изменений в ресурс.

Из формы мы получаем название станции. Изначально нужно проверить, есть ли в базе данных вообще станция с таким названием. Если такая станция есть, то продолжаем работу, елси нет, то нужно вывести предупреждение, что такой станции нет и данные не могут быть предоставлены.

Осуществлять поиск по базе данных удобнее по id станции, а не по названию, поскольку только таблица *Stations* хранит название станции, а все остальные только id. Чтобы получить id станции по ее названию используем как раз основную таблицу *Stations*, котораяхранит в себе основную информацию по каждой станции. Далее нам необходимо обратиться к базе данных.

Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые моделями. Обращаясь к модели, мы получаем записи из нашей базы данных как объект QuerySet [2 – 4].

QuerySet — это итерируемый объект из базы данных, означающий, что он содержит несколько объектов, которые мы можем перебирать / прокручивать. Объект также может использовать фильтры для ограничения результатов [2 – 4].

Каждая модель Django имеет как минимум один менеджер, а менеджер по умолчанию называется objects. Сделать запрос к объекту (QuerySet) можно с помощью менеджера модели [2 – 4].

Ниже приведен фрагмент кода получения id станции по ее названию.

id\_st = Stations.objects.select\_related(**'id'**).filter(NameStation=search\_query)[:1]

search\_query – переменная, которая хранит полученное из формы при помощи метода GET название станции.

Применяем метод filter() к QuerySet, чтобы получить объекты для нужной станции. Методом select\_related достаем именно id.

Получив id станции, обращаемся к базе данных и получаем данные со всех таблиц по интересующей станции.

Ниже приведен пример кода обращения к таблицам в базе данных, содержащим среднемесячные часовые значения суммарной, прямой и диффузной солнечной радиации.

hour\_sum = hour\_sum\_sum\_sr.objects.filter(IdStations=id\_st)  
hour\_straight = hour\_sum\_straight\_sr.objects.filter(IdStations=id\_st)  
hour\_diff = hour\_sum\_diff\_sr.objects.filter(IdStations=id\_st)

Получив данные, в конечном итоге, их нужно отобразить HTML - шаблоне.

return render(request, 'MainApp/SearchResult.html', context= context)

Используем функцию render() для того, чтобы создать HttpResponse, который будет отправлен назад браузеру из нашего представления. Эта функция является «ярлыком», она создаёт HTML-файл, комбинируя указанный HTML-шаблон и некоторые данные для вставки в шаблон (предоставляется в переменной с именем «context») [4].

Переменная *context* содержит в себе слежующие поля:

**Табл.**

Содержимое перемнной context

|  |  |
| --- | --- |
| Название поля | Описание поля |
| res | Хранит объект с информацией по станции |
| res\_mes\_sum | Хранит объект со среднемесячными месячными значениями суммарной слонечной радиации |
| res\_mes\_straight | Хранит объект со среднемесячными месячными значениями прямой слонечной радиации |
| res\_mes\_diff | Хранит объект со среднемесячными месячными значениями диффузной слонечной радиации |
| res\_sut\_sum | Хранит объект со среднемесячными суточными значениями суммарной слонечной радиации |
| res\_sut\_straight | Хранит объект со среднемесячными суточными значениями прямой слонечной радиации |
| res\_sut\_diff | Хранит объект со среднемесячными суточными значениями диффузной слонечной радиации |
| res\_hour\_sum | Хранит объект со среднемесячными часовыми значениями суммарной слонечной радиации |
| res\_hour\_straight | Хранит объект со среднемесячными часовыми значениями прямой слонечной радиации |
| res\_hour\_diff | Хранит объект со среднемесячными часовыми значениями диффузной слонечной радиации |
| albedo | Хранит объект со среднемесячными коэффициентами отражающей способности поверхности – альбедо |
| error | Хранит текст ошибки |

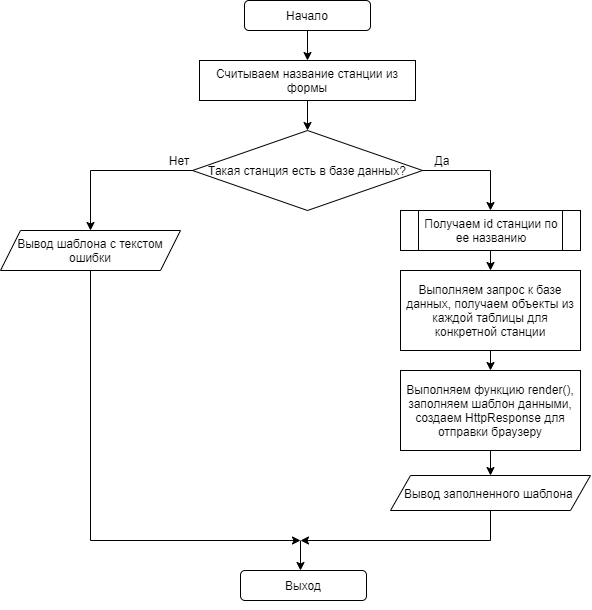


Рис. Алгоритм поиска станции по названию

**Поиск по номеру**

Данный тип поиска практически аналогичен поиску по названию. Отличие состоит в том, что пользователь указывает в поле формы номер станции, что эквивалентно ее id. Таким образом, мы минуем этап поиска id. Также отсутствует проверка на наличие станции в базе данных. Поскольку нам изначально известно общее количество станций, то установив макисимальное и минимальное значение для поля ввода формы, мы производим валидацию данных со стороны клиента. Некорректные данные в принципе уйти на сервер не смогут.

Мы также считываем номер станции из формы. Форма состоит из одного поля, в которое пользовотелю предложено ввести номер станции. Данные передаются на сервер методом GET. На данном этапе мы считаем, что id – это номер введенной станции.

Далее осуществляем запрос к базе данных. Как результат получаем объекты QuerySet со всех таблиц по конкретной станции, отфильтрованные по id.

Полученные данные отображаем в HTML – шаблоне.

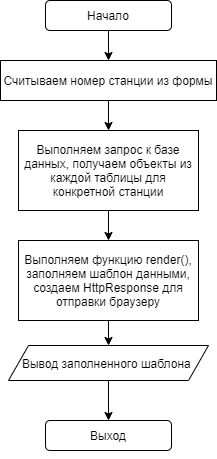


Рис. Алгоритм поиска станции по номеру

**Поиск по географическим координатам**

Поиск по географическим координатам проходит в два этапа: задание пользователем широты и долготы и выбор одной из трех предложенных ближайших к указанной точке станций.

Пользовательская форма для поиска по географическим координатам имеет два поля ввода. Пользователю предолжено указать долготу и широту точки расположения станции. Поля ввода также проверяются на валидность со стороны клиента, следовательно, никаких дополнительных проверок со стороны сервера не требуется.

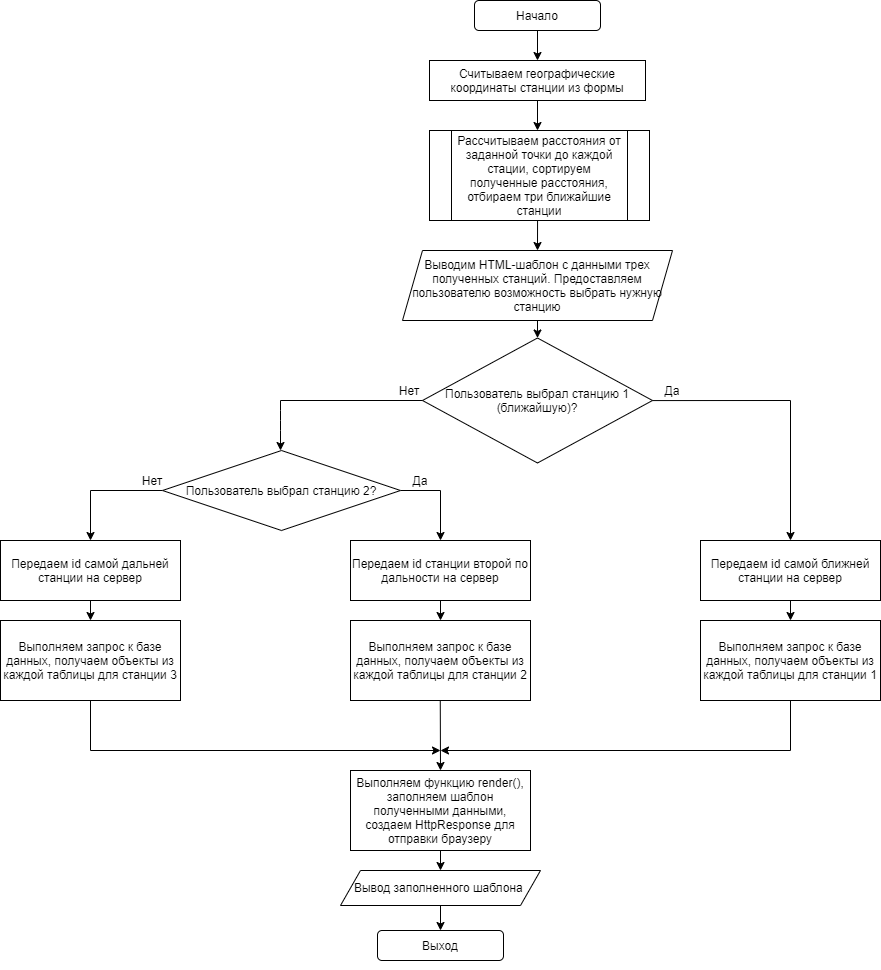
Изначально считываем из формы широту и доготу и передаем на сервер посредством метода GET. Далее рассчитываем расстояния от указанной пользователем точки до каждой станции. Результатом будут три ближайшие станции. Расчет производился при помощи библиотеки «geopy» [6]. Пример кода расчета расстояния показан ниже.

dist = distance.distance(coords\_1, coords\_2).km

Функция distance.distance рассчитывает расстояние между двумя точками, используя геодезическое расстояние. Геодезическое расстояние - это кратчайшее расстояние на поверхности эллипсоидальной модели Земли [6]. Функция принимает два параметра: географические координаты заданной точки (coords\_1) и географические координаты станции (coords\_2). Параметр .km означает, что расстояния будут выражаться в километрах. При рассчете не учитывается высота расположение точек.

Расстояния рассчитываются для каждой станции по порядку, а результат заносится в массив по индексу, который соответствует номеру станции. Далее сортируем полученный массив по значению расстояния в порядке возрастания. Нас интересуют индексы первых трех элементов массива, поскольку это и будут три ближайшие станции.

Далее отображаем HTML – шаблон, на котором выводим по отобранным трем станциям информацию в виде формы, для предоставления пользователю возможности выбора нужной станции. В зависимости от того, какой выбор сделает пользователь, нажав на кнопку, соответствующую одной из трех представленных станций, на сервер отправится id этой станции. На сервере при помощи GET запроса мы получаем id и делаем запрос к базе данных для отбора информации. Как результат обращения получаем объекты QuerySet, отфильтрованные по id. Полученные данные отображаем в HTML – шаблоне.



**5. Разработка алгоритма интерполяции данных**

База данных содержит в себе среднемесячные часовые, суточные и месячные суммы прихода солнечной радиации на горизонтальную поверхность. Однако в ряде случаев предоставить данные пользователю необходимо не в виде средних заначений за месяц, а за каждый час месяца. Исходя из этого, появляется потребность в интерполировании данных.

Интерполировать данные предстоит двумя способами:

1. Интерполяция по принципу – “День сурка”;
2. Интерполяция по принципу – “Линейная”;

**3.1. Интерполяция “День сурка”**

**3.1.1.Описание принципа итерполяции типа “День сурка”**

Суть интерполяции по принципу «День сурка» состоит в том, что имеющееся среднемесячное значение прихода солнечной радиации для некоторого месяца мы просто повторяем для каждого дня данного месяца. Никакой дополнительной обработки над данными не производится. Считается, что каждый день месяца было именно такое значение. Данный подход достаточно сильно упрощает процесс интерполяции, но при этом точность получаемой информации падает.

**3.1.2. Описание алгоритма интерполяции**

На вход алгоритма поступает QuerySet, который мы фильтруем при помощи метода filter(), отбирая данные для конкретной станции. Фильтрация идет по id станции, который мы получили ранее при поиске.

Объект QuerySet будет содержать данные для конкретной станции:

*id, Month\_1, Month\_2, Month\_3, Month\_4, Month\_5, Month\_6, Month\_7, Month\_8, Month\_9, Month\_10, Month\_11, Month\_12*.

Для перебора данных необходимо сериализовать объект. Сериализация — это преобразование списка или словаря в JSON-строку.

*object\_list* – это объект, содержащий в себе сериализованные данные, содержащиеся в в QuerySet, но в другом формате представления.

Сериализация — это преобразование списка или словаря в JSON-строку.

Формат данных, хранящихся в object\_list:

{'model': 'hour\_sum\_sum\_sr', 'pk': 121, 'fields': {'IdStations': 6, 'IdHour': 1, 'Month\_1': Decimal('0.00'), 'Month\_2': Decimal('0.00'), 'Month\_3': Decimal('0.00'), 'Month\_4': Decimal('0.00'), 'Month\_5': Decimal('0.00'), 'Month\_6': Decimal('0.00'), 'Month\_7': Decimal('0.00'), 'Month\_8': Decimal('0.00'), 'Month\_9': Decimal('0.00'), 'Month\_10': Decimal('0.00'), 'Month\_11': Decimal('0.00'), 'Month\_12': Decimal('0.00')}}

Как результат получаем словарь – коллекцию объектов с доступом по ключу, который удобно итерировать.

Далее необходимо заполнить массивы данных, с помощью которых мы будем производить интерполяцию. Тип хранимых данных и назначение каждого массива описаны в Табл. 6.

**Табл. 6**

**Описание массивов данных**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип хранимых данных | Описание переменной |
| a11 | Float(4,2) | Массив, храянящий значения за 15-ое января, за 24 часа |
| a12 | Float(4,2) | Массив, храянящий значения за 15-ое февраля, за 24 часа |
| a13 | Float(4,2) | Массив, храянящий значения за 15-ое марта, за 24 часа |
| a14 | Float(4,2) | Массив, храянящий значения за 15-ое апреля, за 24 часа |
| a15 | Float(4,2) | Массив, храянящий значения за 15-ое мая, за 24 часа |
| a16 | Float(4,2) | Массив, храянящий значения за 15-ое июня, за 24 часа |
| a17 | Float(4,2) | Массив, храянящий значения за 15-ое июля, за 24 часа |
| a18 | Float(4,2) | Массив, храянящий значения за 15-ое августа, за 24 часа |
| a19 | Float(4,2) | Массив, храянящий значения за 15-ое сентября, за 24 часа |
| a110 | Float(4,2) | Массив, храянящий значения за 15-ое октября, за 24 часа |
| a111 | Float(4,2) | Массив, храянящий значения за 15-ое ноября, за 24 часа |
| a112 | Float(4,2) | Массив, храянящий значения за 15-ое декабря, за 24 часа |
| а | Float(4,2) | Выходной массив, содержащий всю выходную последовательность |

Берем значения из object\_list по ключу fields. Значению по ключу fields соответствует также словарь, который хранит в себе данные о станции и месячные значения для этой станции. Основная работа будет производится именно с этими данными.

В цикле перебераем поля каждого месяца по field\_name (ключ - название поля) и записываем значение field\_value в массив, соответствующий данному месяцу. Пример кода представлен в Приложении 1.

После того, как массивы заполняются исходными данными, начинается процесс интерполяции. Суть состоит в том, что формируется выходной массив как объеденение всех массивов, каждый из которых повторяется некоторое количество раз, в соответствии с количеством дней в месяце. Пример кода получения выходного массива показан ниже.

a = a11\*31 + a12\*28 + a13\*31 + a14\*30 + a15\*31 + a16\*30 + a17\*31 + a18\*31 + a19\*30 + a110\*31 + a111\*30 + a112\*31

Общее количество элементов в массиве а = 8760, что соответствует количеству часов в году.

**3.2. Интерполяция “Линейная”**

**3.2.1 Описание принципа интерполяции типа “Линейная”**

«Линейная интерполяция» в данном случае отличается от общепринятого понятия, поскольку выполняется она совсем другим способом. Описанный ниже процесс называется линейной интерполяцией в солнечной энергетике.

Изначально мы имеем только средние за месяц значения прихода солнечной радиации. Затем мы предполагаем, что эти значения приходятся на конкретное число месяца – на 15-ое. Теперь мы имеем точки, приходящиеся на середину каждого месяца, которые нам нужно «соединить».

Для того чтобы соединить две соседние точки, мы должны найти шаг, с которым мы будем идти. Находим мы его по формуле:

, где (1)

a – значение в месяце;

b – значение в месяце, который идет следующим за «а»;

n – количество дней между 15-ым числом одного месяца и 15-ым числом следующего;

Здесь необходимо учитывать, что от месяца к месяцу n будет различным, поскольку в месяце может быть и 30, и 31, и 28 дней (считается, что год невисокосный). От этого возникают дополнительные условия.

Также необходимо найти разность между этими двумя значениями, чтобы понять, увеличивается ли приход солнечной радиации в следующем месяце или наоборот уменьшается.

* Если разность меньше нуля, то делаем вывод, что приход солнечной радиации уменьшается и, следовательно, необходимо уменьшать значение «a».
* Если разность больше нуля, то приход солнечной радиации, наоборот, увеличивается и, следовательно, нужно увеличивать «a».
* Если разность равна ноль, то приход солнечной радиации не изменился, поэтому значения на протяжении всего интервала будут одинаковыми.

Процесс интерполяции – это не что иное, как последовательное увеличение (уменьшение) значения «a» на полученный шаг.

В идеальном случае при таком последовательном увеличении (уменьшении) величины «a» на шаге n мы должны получить значение равное исходному значению «b».

Такие действия повторяются для каждого часа месяца.

Таким образом, мы «восстанавливаем» неизвестные нам значения, находящиеся в промежутке между 15-ым числом одного месяца и 15-ым числом следующего.

**3.2.2. Описание алгоритма интерполяции**

Исходные данные аналогичны интерполяции по принципу «День сурка», то есть работаем с массивами: *a11, a12, a13, a14, a15, a16, a17, a18, a19, a110, a111, a112*. Массивы заполнены среднемесячными значениями прихода солнечной радиации для конкретного месяца.

Для каждого промежутка (от 15-го числа одного месяца до 15 числа следующего месяца) мы находим разность и запоминаем ее в переменной *raz*. Сравниваем ее с нулем, чтобы оценить увеличивается ли приход солнечной радиации или нет. Находим шаг дискретизации по формуле (1) и запоминаем в переменной *delt*. Для каждого промежутка знаменатель в формуле будет меняться, поскольку это зависит от количества дней в месяце. Сохраняем в переменную *nach -* начало промежутка. Далее в зависимости от того, положительна разность или отрицательна, мы в цикле, длительность которого зависит от количества дней между двумя точками, либо прибавляем *delt*, либо вычитаем из переменной *nach*. Если разность равна нулю, то прибавляем *delt* к *nach*, значение от этого не изменится. Каждое полученное значение записываем в выходные массивы. Тип вышеупомянутых перемнных и их значения описаны в Табл. 7.

**Табл. 7**

**Описание переменных, используемых при интерполировании**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип переменной | Значение переменной |
| raz | Float(4,2) | Разность между начальным и конечным значением |
| nach | Float(4,2) | Значение начала периода |
| delt | Float(4,2) | Шаг дискретизации |

Процесс повторяется для каждого промежутка 24 раза, по количеству часов в сутках.

Таким образом, заполняются выходные массивы значениями, полученными в результате интерполяции. Тип хранимых данных и назначение массивов описываются в Табл. 8.

**Табл. 8**

**Описание выходных массивов**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип хранимых данных | Значение переменной |
| a21 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го января по 15-ое февраля |
| a22 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го февраля по 15-ое марта |
| a23 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го марта по 15-ое апреля |
| a24 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го апреля по 15-ое мая |
| a25 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го мая по 15-ое июня |
| a26 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го июня по 15-ое июля |
| a27 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го июля по 15-ое августа |
| a28 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го августа по 15-ое сентября |
| a29 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го сентября по 15-ое октября |
| a210 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го октября по 15-ое ноября |
| a211 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го ноября по 15-ое декабря |
| a212 | Float(4,2) | Массив, хранящий интерполированные значения для промежутка с 15-го декабря по 15-ое января |

Необходимо также учесть переход от 15-го декабря к 15-му января. Получается, что часть интерполированных значений, соответствующих периоду с 15-го декабря по 31-го декабря, должны находиться в конце выходной последовательности, а часть, соответствующая периоду с 1-го января по 15-ое января, наоборот находиться в начале. Таким образом, мы делим массив *а212* на две части и, при формировании выходного массива, помещаем части на соответствующие места.

Табл. 9 содержит описание переменных, которые будут использоваться для правильного формирования выходной последовательности.

**Табл. 9**

**Описание переменных для формирования выходной последовательности**

|  |  |  |
| --- | --- | --- |
| Название переменной | Тип переменной | Значение переменной |
| begin | Float(4,2) | Массив, содержащий часть массива а212, которая соответствует периоду с 1-го января по 15-января |
| end | Float(4,2) | Массив, содержащий часть массива а212, которая соответствует периоду с 15-го декабря по 31-декабря |
| a | Float(4,2) | Выходной массив, в котором мы собираем данные за все месяцы |

На Рис. 6 представлена схема алгоритма интерполяции данных.

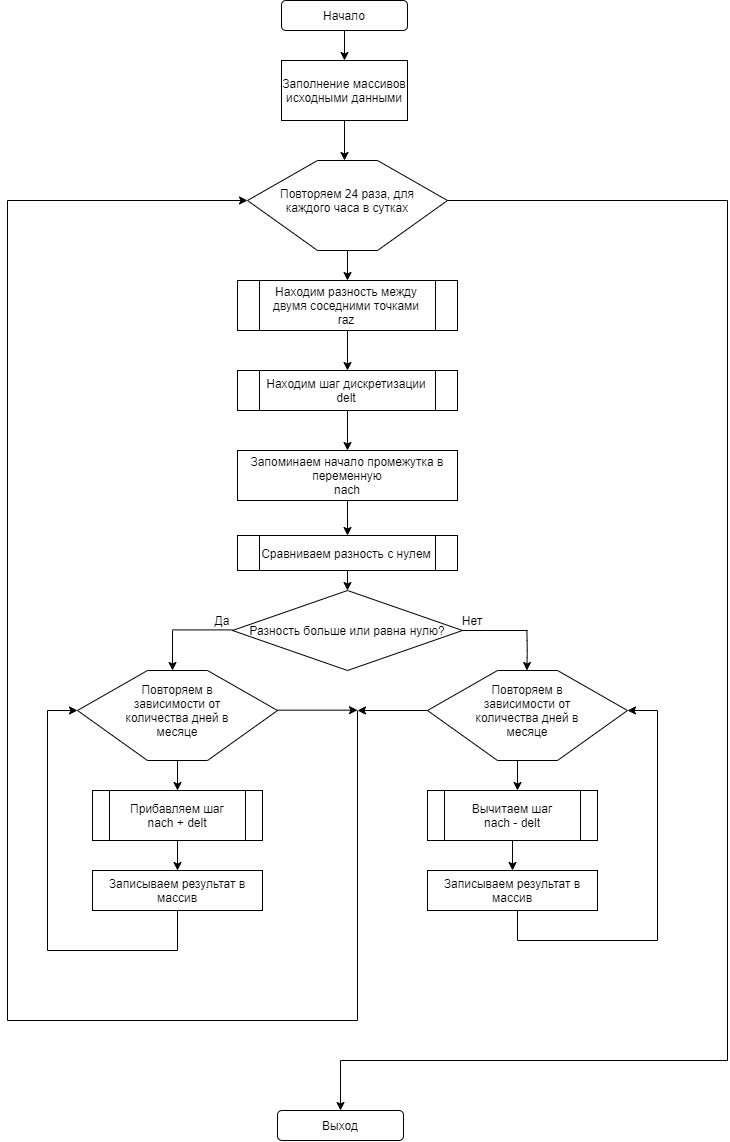


Рис. 6 Схема алгоритма интерполяции

**4. Экспорт в csv**

В данном разделе описывается алгоритм экспорта данных для выбранной станции в формате csv.

Всего экспортируется 5 таблиц: таблица альбедо, таблицы среднемесячных часовых сумм суммарной, прямой и диффузной солнечной радиации, а так же таблица, формируемая в результате интерполяции.

Первые четыре экспортируются в том же виде, в каком хранятся в базе данных, то есть в виде среднемесячных значений.

Последняя таблица формируется в зависимости от заданных пользователем параметров: период, за который необходимо получить данные, тип интерполяции и шаг дискретизации.

Есть два варианта указания периода:

* за весь период наблюдений;
* за указанный интервал;

Два типа интерполяции:

* День сурка;
* Линейная;

Два варианта шага дискретизации:

* 1 час;
* 2 часа;

В зависимости от сочетания значений этих трех входных параметров мы получаем на выходе некоторую таблицу, которую и направляем на экспорт.

**4.2. Описание алгоритма экспорта интерполированных данных**

На входе алгоритма имеем 3 параметра, которые получаем при помощи GET запроса от пользовательской формы.

1. *Period* – период за который мы хотим получить данные.

Возможны два варианта:

*All\_period* – данные берутся за весь период наблюдения (1 год);

*Part\_period*  – данные берутся за указанный интервал;

Для задания интервала указываются 6 параметров:

*Mes\_begin* – месяц начала интервала (обязательный параметр);

*Day\_begin*  – день начала интервала;

*Hour\_begin* – час начала интервала;

*Mes\_end* – месяц конца интервала (обязательный параметр);

*Day\_end* – день конца интервала;  
*Hour\_end* – час конца интервала;

1. *Type* – тип интерполяции, который мы хотим применить к нашим исходным данным.

Возможны два варианта:

*Type\_surok* – интерполяция по принципу «День сурка»;

*Type\_lin* – интерполяция по принципу «Линейная интерполяция»;

1. *Hour* – шаг дискретизации, с которым мы хотим интерполировать данные.

Возможны два варианта:

*1\_Hour* – шаг дискретизации 1 час, то есть берем каждое значение;

*2\_Hour* – шаг дискретизации 2 часа, то есть берем каждое второе значение;

На Рис. 8. представлен вид пользовательской формы приложения.

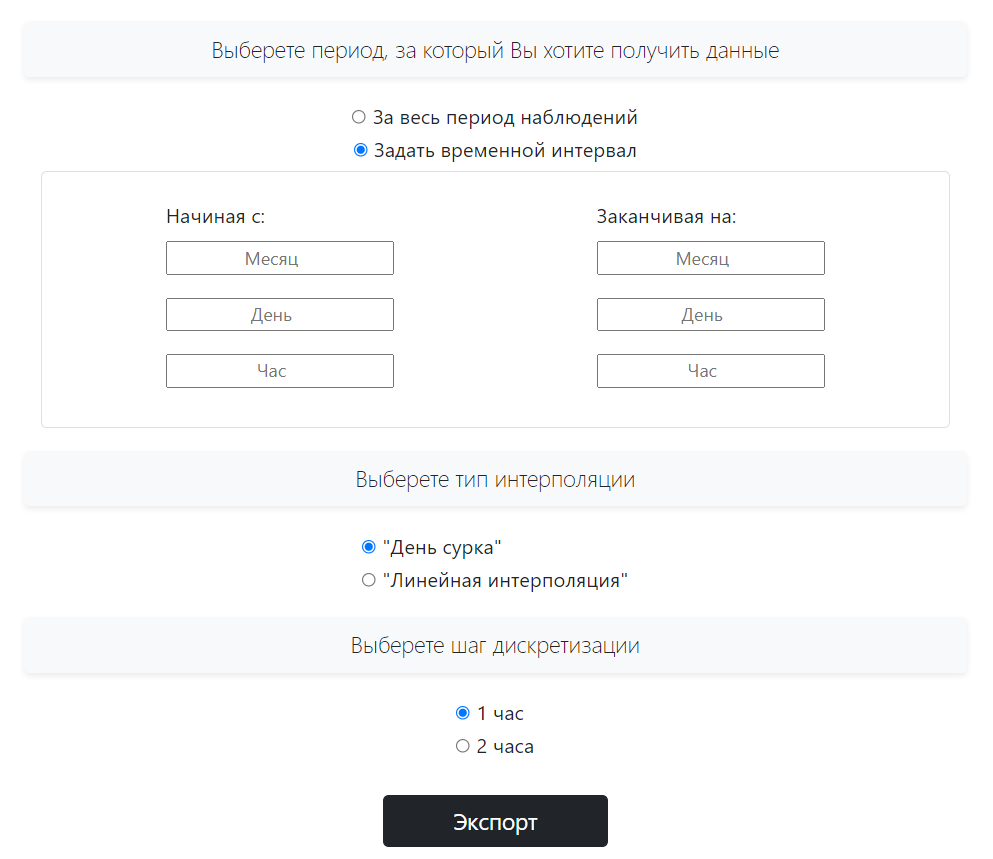


Рис. 8 Вид пользовательской формы

Все три параметра должны быть обязательно выбраны. Это достигается использованием настраиваемой логики проверки для полей. Обязательные для заполнения поля имеют атрибут required. Поэтому, если пользователь не выберет период, тип интерполяции или шаг дискретизации, то фреймворк выдаст предупреждение, в котором попросит выбрать один из вариантов.

Обязательно для заполнения поле – Месяц. Если мы не укажем месяц, то получим неопределенность в запрашиваемых данных. Однако день и час можно не указать, тогда просто устанавливается по умолчанию:

* начало периода:

*Day\_begin* = 1 (первый день месяца);

*Hour\_begin* = 1 (первый час дня);

* конец периода:

*Day\_end* = 31/30/28 (последний день месяца, а в месяце может быть либо 31, либо 30, либо 28 дней);  
*Hour\_end* = 24 (последний час дня);

В соответствии со значением полученного параметра *Type* обрабатываем массивы, содержащие интерполированные данные часовых сумм суммарной, прямой и диффузной солнечной радиации, а также альбедо.

Формируем массивы с индексацией: *id, HOY, DOY, Month, Day, Hour,* в соответсвии с предоставленным институтом ГВИЭ форматом выходного файла.

Далее в цикле последовательно подаем все этим массивы на экспорт. В зависимости от заданных параметров берутся определенные элементы массивов в соответствии с индексом.

Если указан шаг дискретизации 1 час, то берутся все значения подряд, если 2 часа, то каждое второе (кроме id, этот столбец должен всегда содержать непрерывную последовательность).

Если указан параметр, указывающий, что нам нужны данные за весь период наблюдений, то мы считываем все массивы целиком. Если нужны данные за какой-то период, то мы, в соответствии с указанными значениями начала и конца периода, высчитываем индекс, с которого нужно начать читать данные, а также индекс, на котором мы должны остановиться.

На Рис. 9 представлена структура алгоритма экспорта данных.

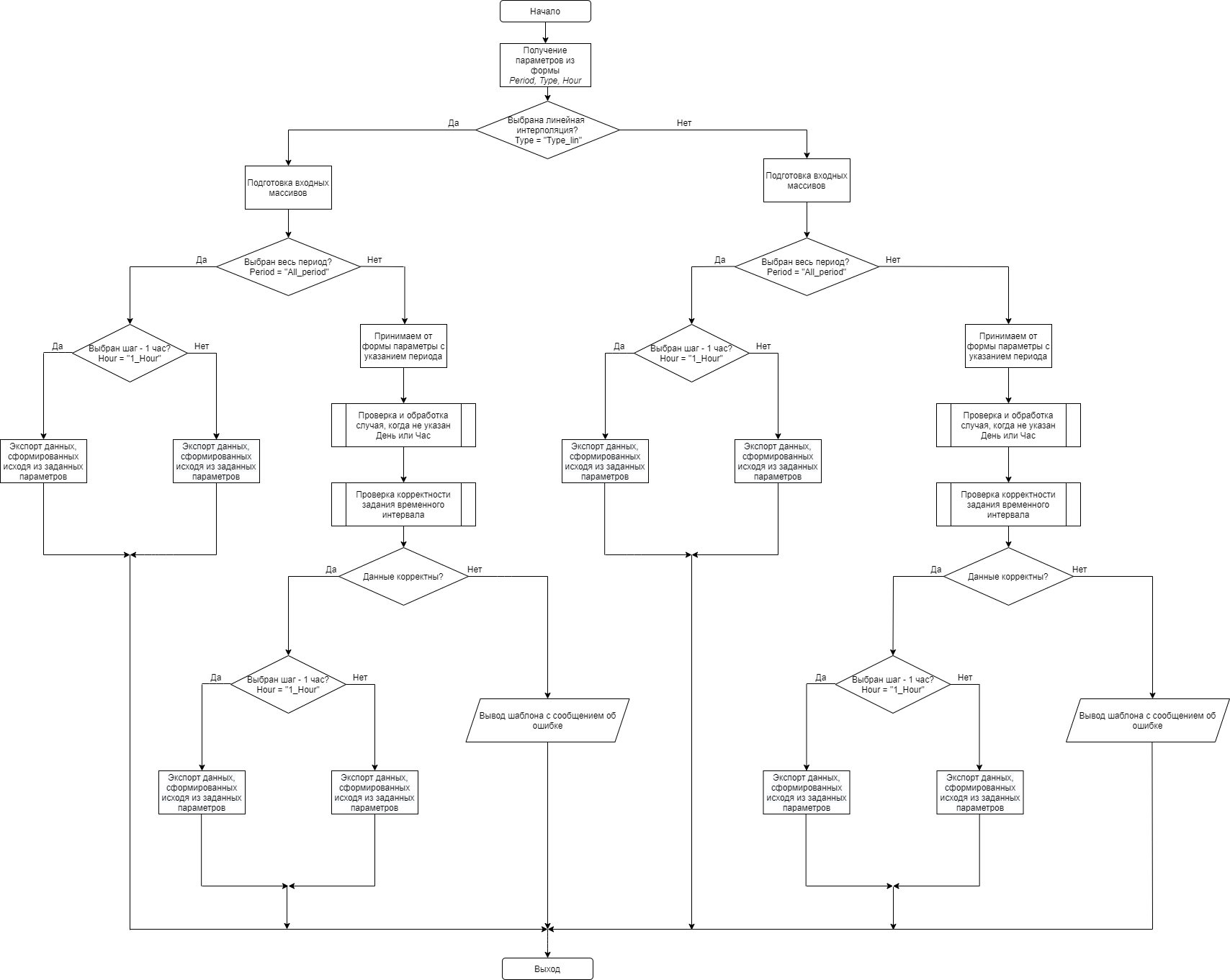


Рис. 9 Схема алгоритма экспорта данных

Для реализации экспорта использовалась библиотека «django-import-export» [1 – 3]. Сначала создается объект HttpResponse с соответствующим заголовком CSV. Это сообщает браузерам, что документ является файлом CSV, а не файлом HTML. Далее подключаемся к API генерации CSV. Передаем ответ из начального аргумента в csv.writer. Определяем возвращаемую информацию, режим вложения и имя файла. Записываем в файл заголовок, а затем построчно данные из массивов. Пример кода реализации экспорта данных показана ниже.

response = HttpResponse(content\_type='text/csv')  
writer = csv.writer(response)  
response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
  
for i in range(8760):  
 writer.writerow([i, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])

Названиям столбцов в заголовке соответствуют следующие значения:

id – индексация

*HOY* – час в году;

*DOY* – день в году;

*Month* – месяц;

*Day* – день;

*Hour* – час;

*GHI* – часовые суммы суммарной солнечной радиации;

*DHI* – часовые суммы диффузной солнечной радиации;

*DirectHI* – часовые суммы прямой солнечной радиации;

*Albedo* – коэффициенты отражающей способности – альбедо;

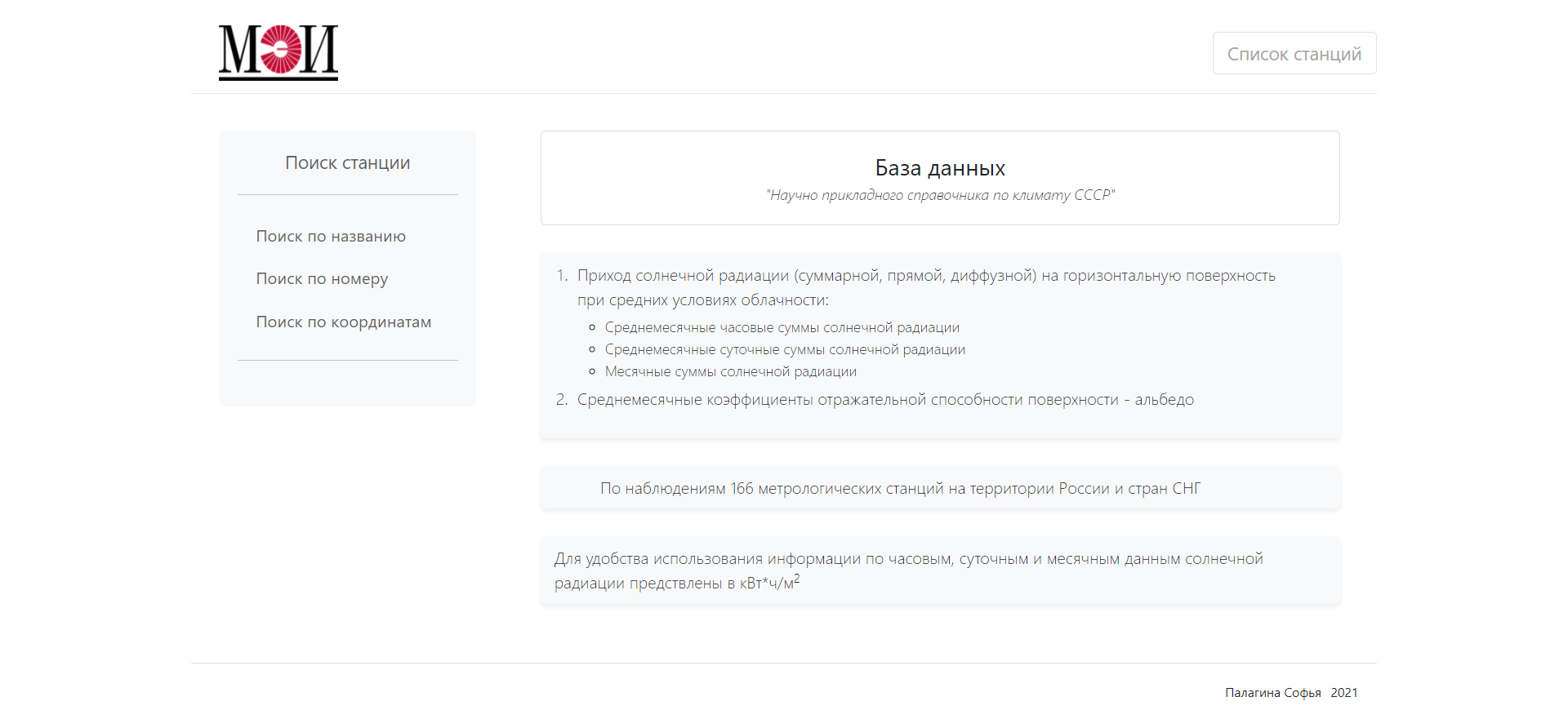
**Разработка пользовательского интерфейса**

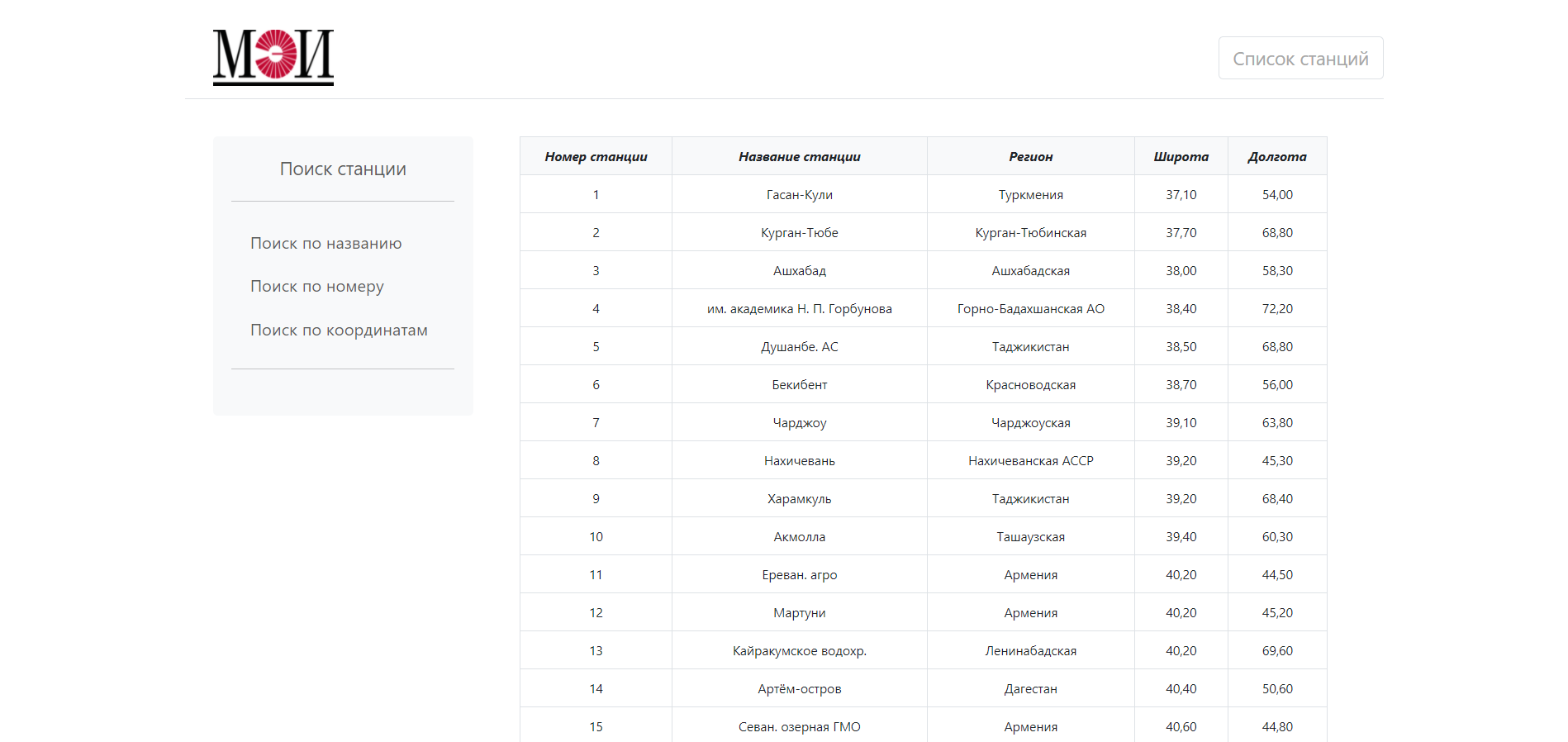
В результате выполнения данной работы был разработан пользовательский интерфейс. Основной задачей было сделать его максимально удобным и интуитивно понятным. Разрабатывался он на языке HTML 5 с помощью языка CSS 3. Предусматривалась адаптивность приложения под различные размеры дисплея, такие как планшет (10 - 11 дюймов) и смартфон (5 - 6 дюймов).

Отображаемые шаблоны содержат 4 основные области: заголовочная часть, навигационное меню, контентная часть и футер сайта. Навигационное меню, заголовочная часть и футер остаются постоянными на всех страницах приложения.

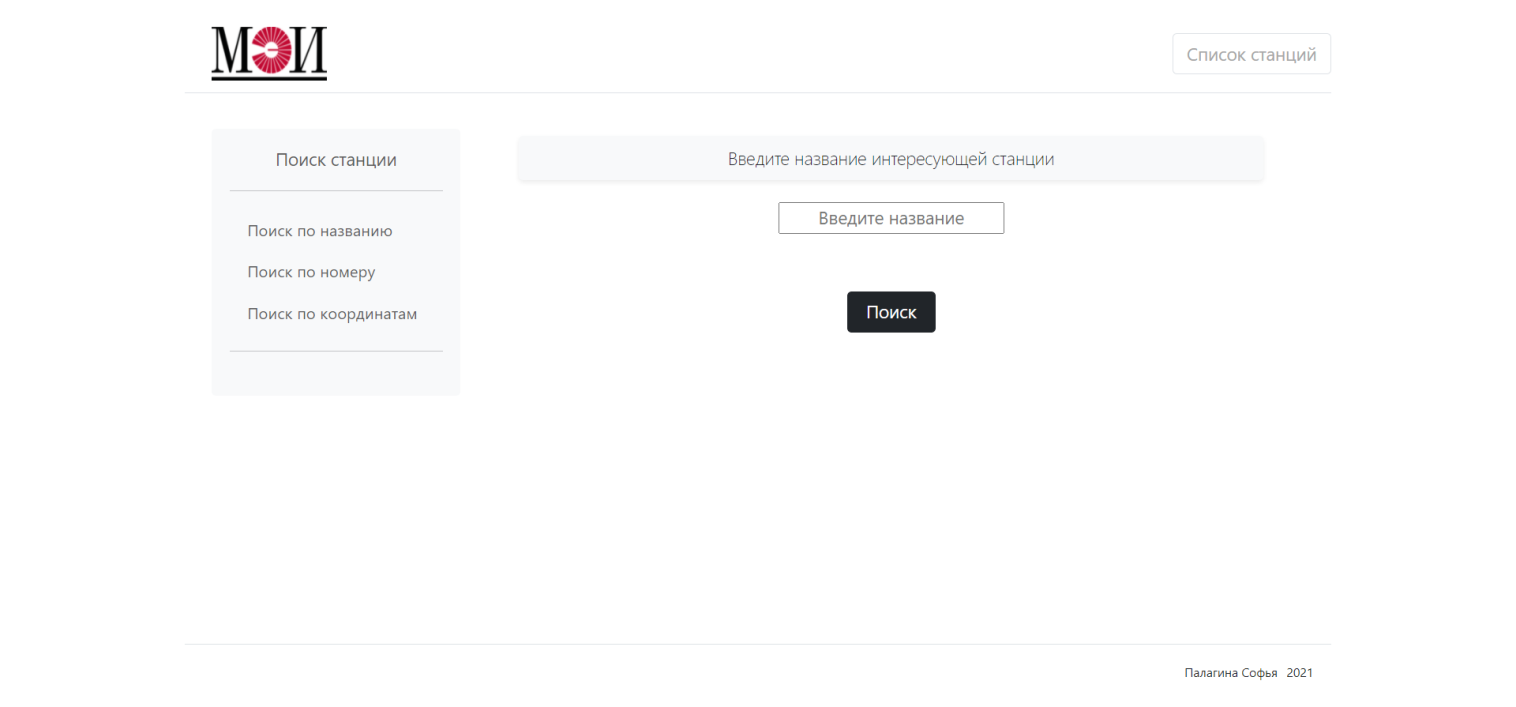
Навигационное меню находится в левой части экрана и содержит ссылки на страницы поиска: «Поиск по названию», «Поиск по номеру», «Поиск по координатам». Рядом с навигационным меню находится область контента, в которой отражается информация в зависимости от выбранного раздела. Шапка сайта содержит кнопку «Список станций», при нажатии на неё отображается таблица, содержащая основную информацию по всем станциям, ее интерфейс показан на Рис. .

Главная страница приложения содержит в контентной части информацию о данных, содержащихся в базе. Интерфейс главной страницы показан на Рис. .

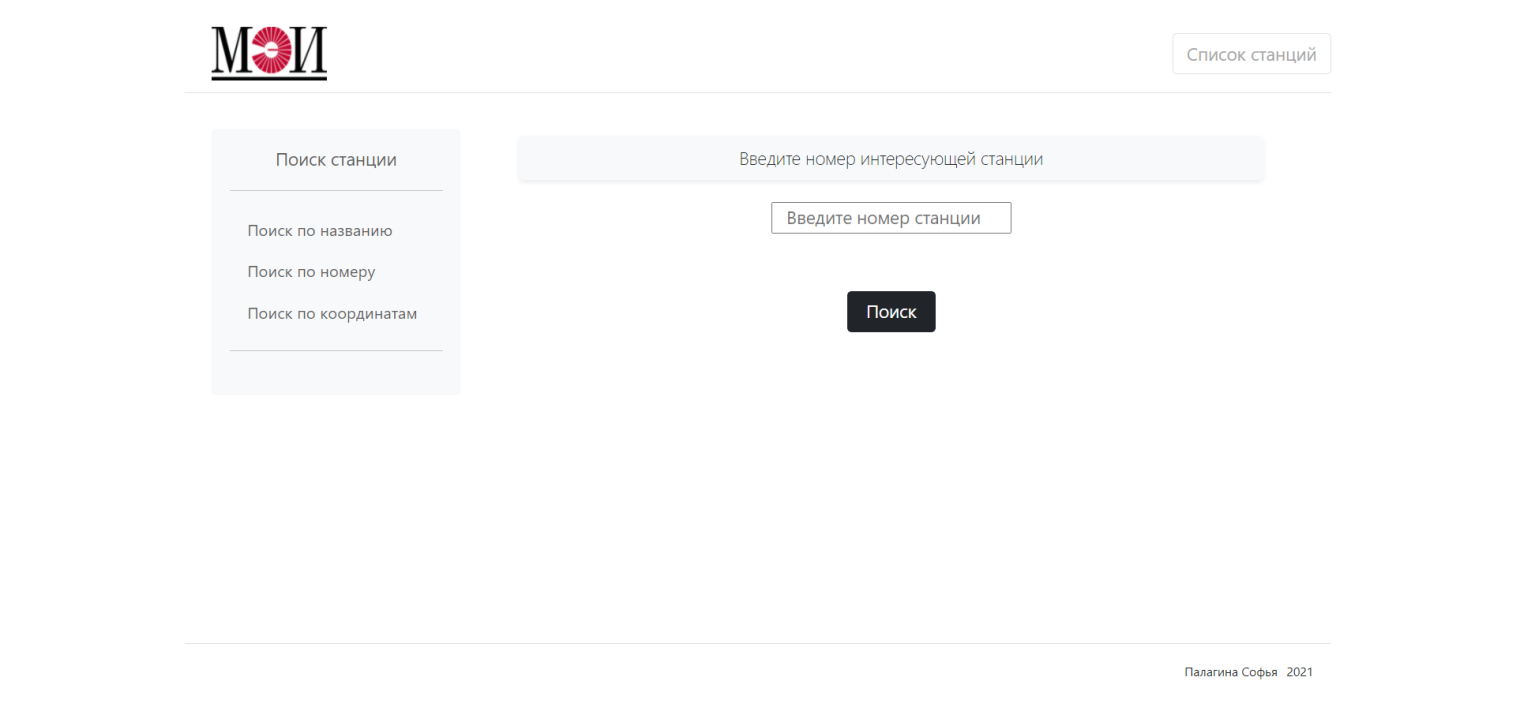


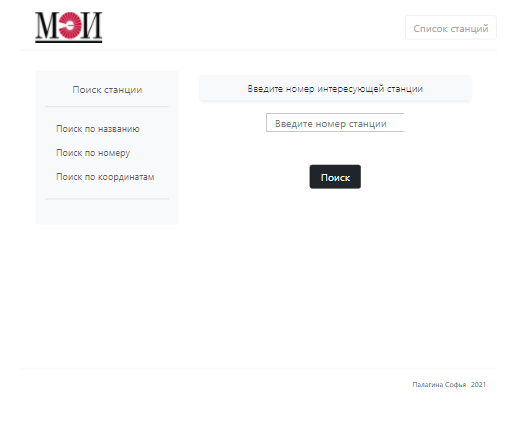


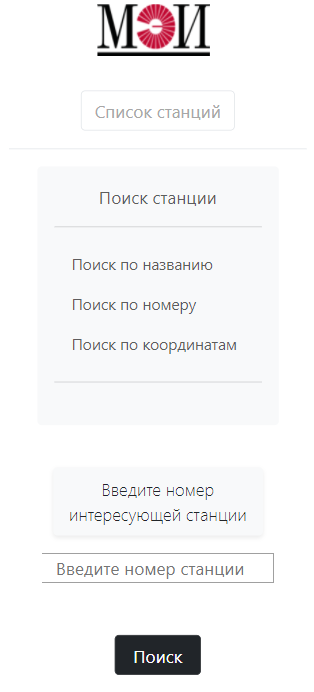
При выборе «Поиск по названию» происходит переход на страницу, где находится поле ввода. В это поле предложено ввести название интересующей станции. После ввода необходимо нажать на кнопку «Поиск», если поиск пройдет удачно, то произойдет переход на страницу, содержащую данные по выбранной станции, а если же нет, то на странице выведется сообщение об ошибке. Интерфейс страницы поиска по названию представлен на Рис. .



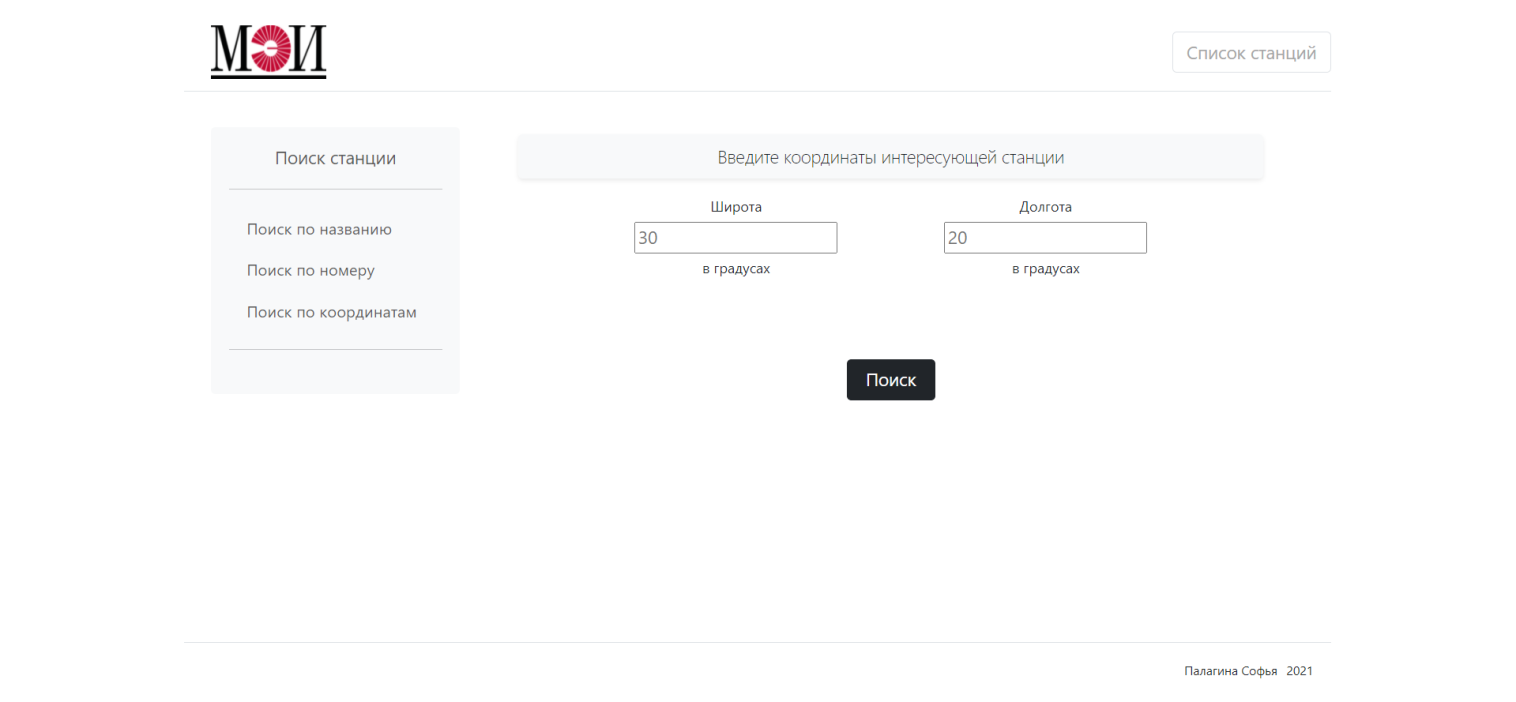
Интерфейс страницы, которая отображается при выборе «Поиск по номеру», аналогичен предыдущей, только в поле ввода необходимо ввести номер станции. Интерфейс страницы «Поиск по номеру» представлен на Рис. , , .



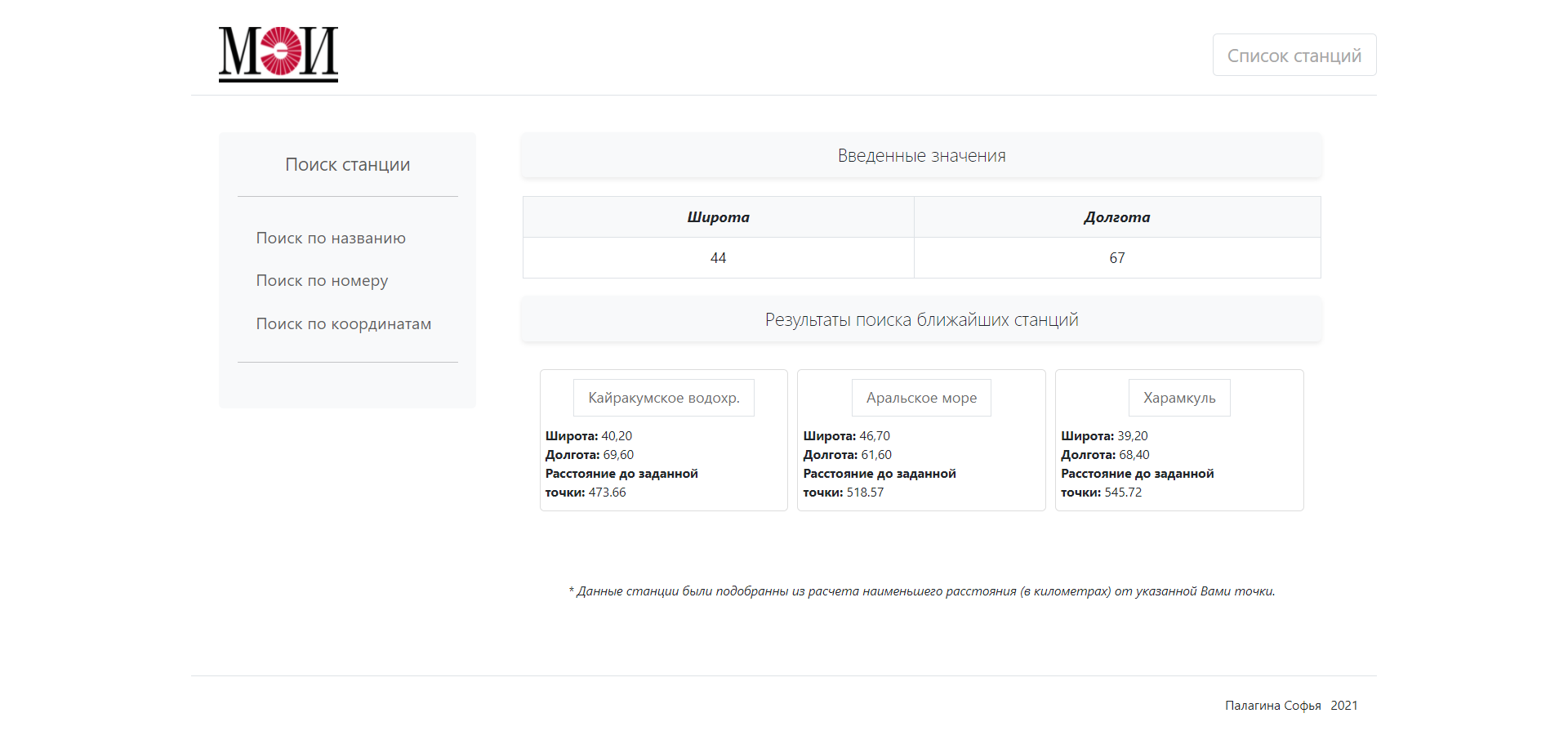


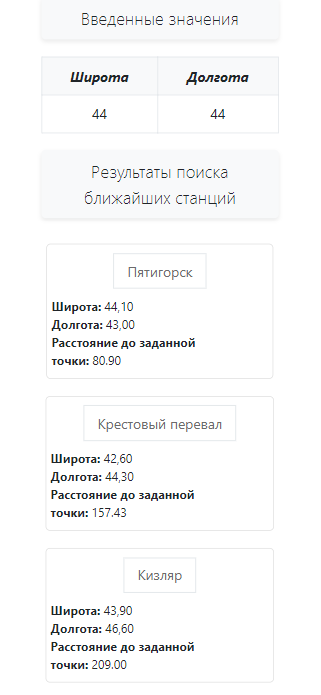


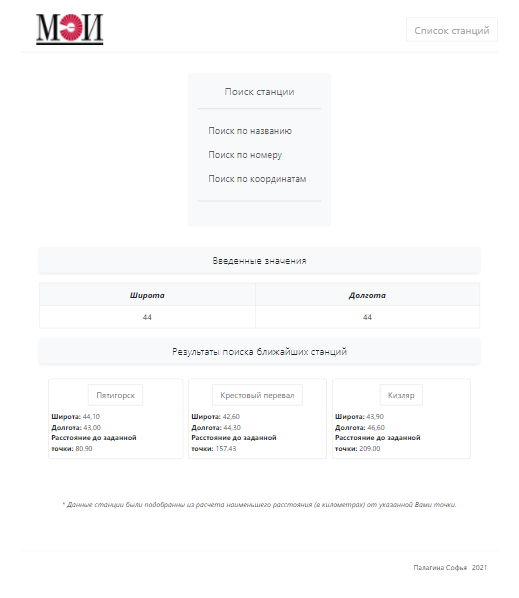
При выборе «Поиск по координатам» происходит переод на страницу, содержащую в контентной области два поля ввода. В первое поле необходимо ввести широту, во второе долготу искомой точки. После заполнения полей необходимо нажать на кнопку «Поиск». Интерфес страницы «Поиск по координатам» представлен на Рис. .



В результате поиска по географическим координатам происходит переход на страницу, где отображаются сначала введенные параметры, а затем три блока, содержащие информацию по ближайшим станциям, а именно широту и долготу ближайшей найденной станции, и расстояние от найденной станции до указанной пользователем точки. Исходя из представленной информации, пользователь модет сделать выбор, нажав на кнопку с названием станции. В результате выбора произойдет переход на страницу, содержащую данные по выбранной станции. Вид пользовательского интерфейса представлен на Рис. , и .



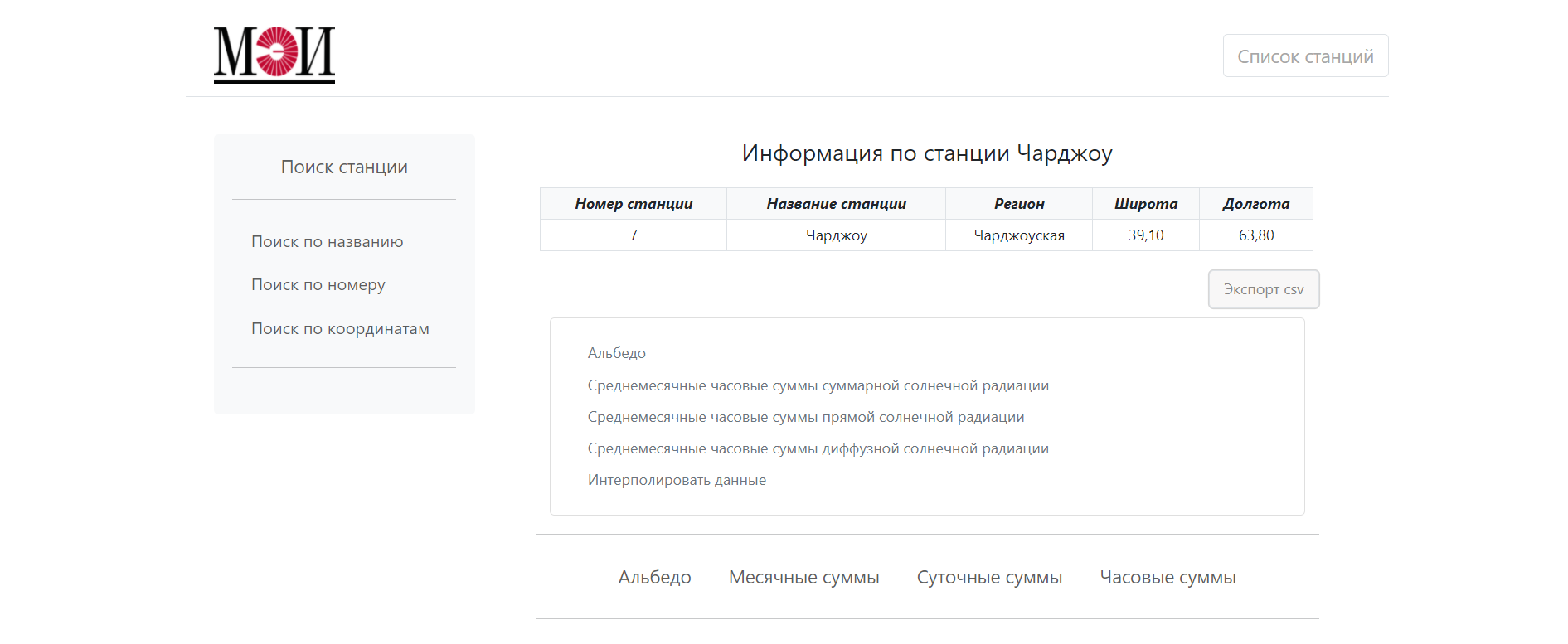


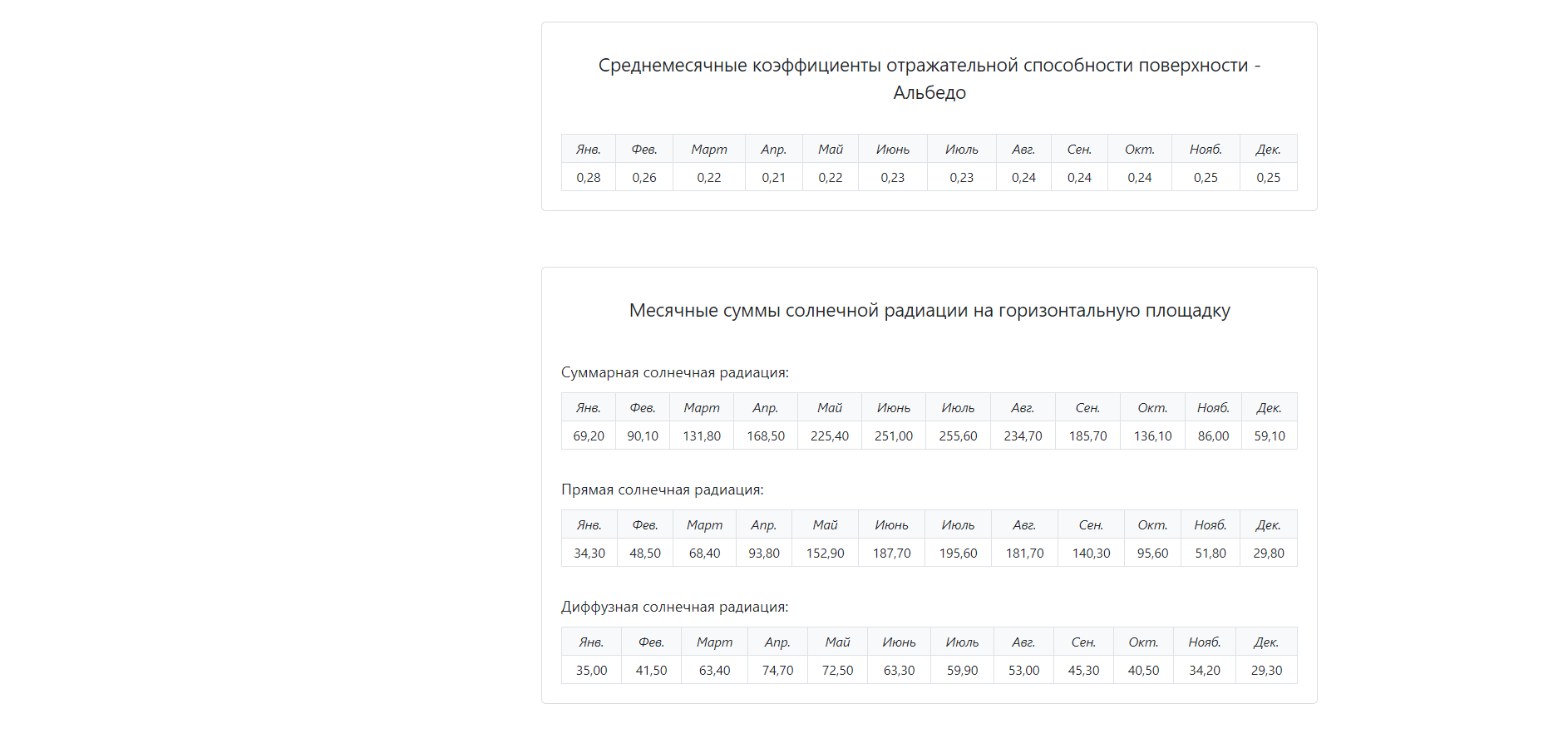


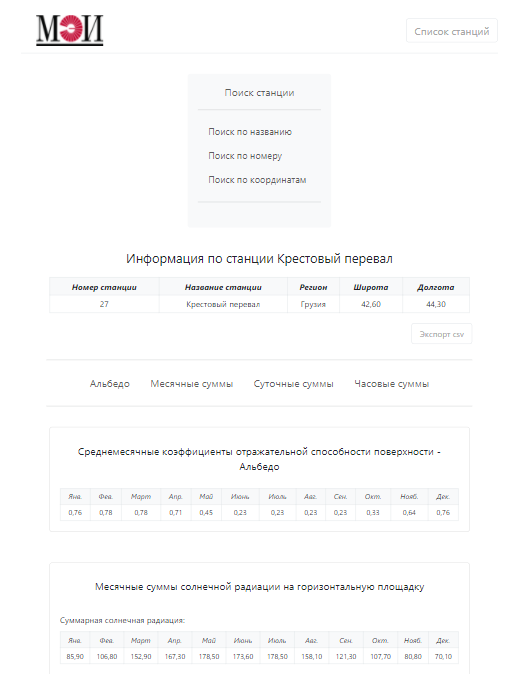
После выбора станции пользователь попадает на страницу, содержащую информацию по станции, Сначала отображается название станции и основная информация по ней. Далее располагается кнопка «Экспорт csv», при нажатии на нее появляется выпадающее меню. Данное меню содержит следующие поля: «Альбедо», «Средние часовые суммы суммарной солнечной радиации», «Средние часовые суммы прямой солнечной радиации», «Средние часовые суммы диффузной солнечной радиации» и «Интерполировать данные». При нажатии на первые четырех ссылки происходит скачивание файлов в формате csv, которые содержат исходные необработанные таблицы: альбедо, средние часовые суммы суммарной солнечной радиации, средние часовые суммы прямой солнечной радиации, средние часовые суммы диффузной солнечной радиации.

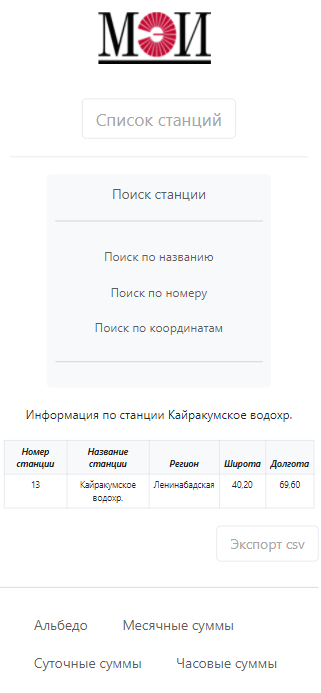
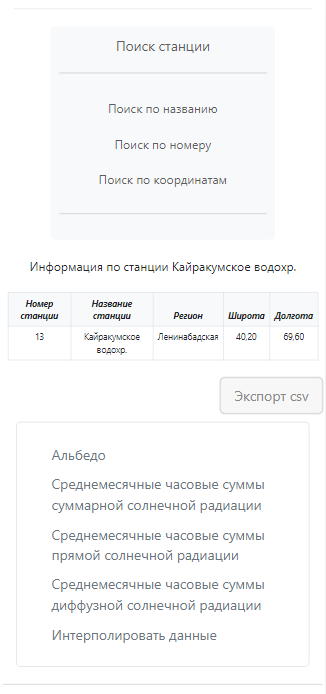
При нажатии на кнопку «Интерполировать данные» происходит переход на страницу для задания параметров интерполяции. Далее идет небольшое навигационное меню, при нажатии на соответсвующую ссылку происходит прокрутка экрана на нужную область страницы, это, своего рода, закладка для быстрого перемещения по странице с целью поиска, называемая якорь сайта.

Далее непосредственно идут сами данные в виде таблиц, которые пользователь может просматривать. Интерфейс страницы представлен на Рис.







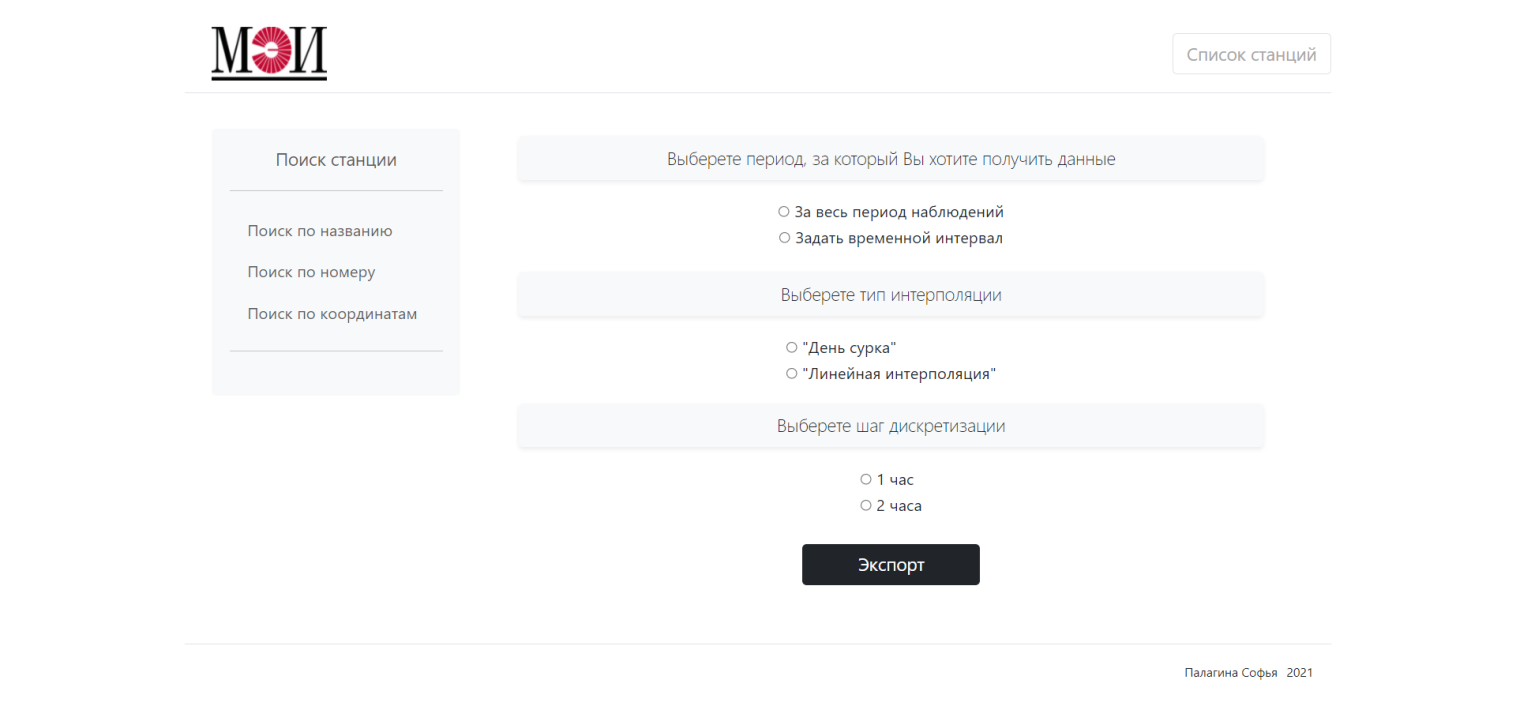


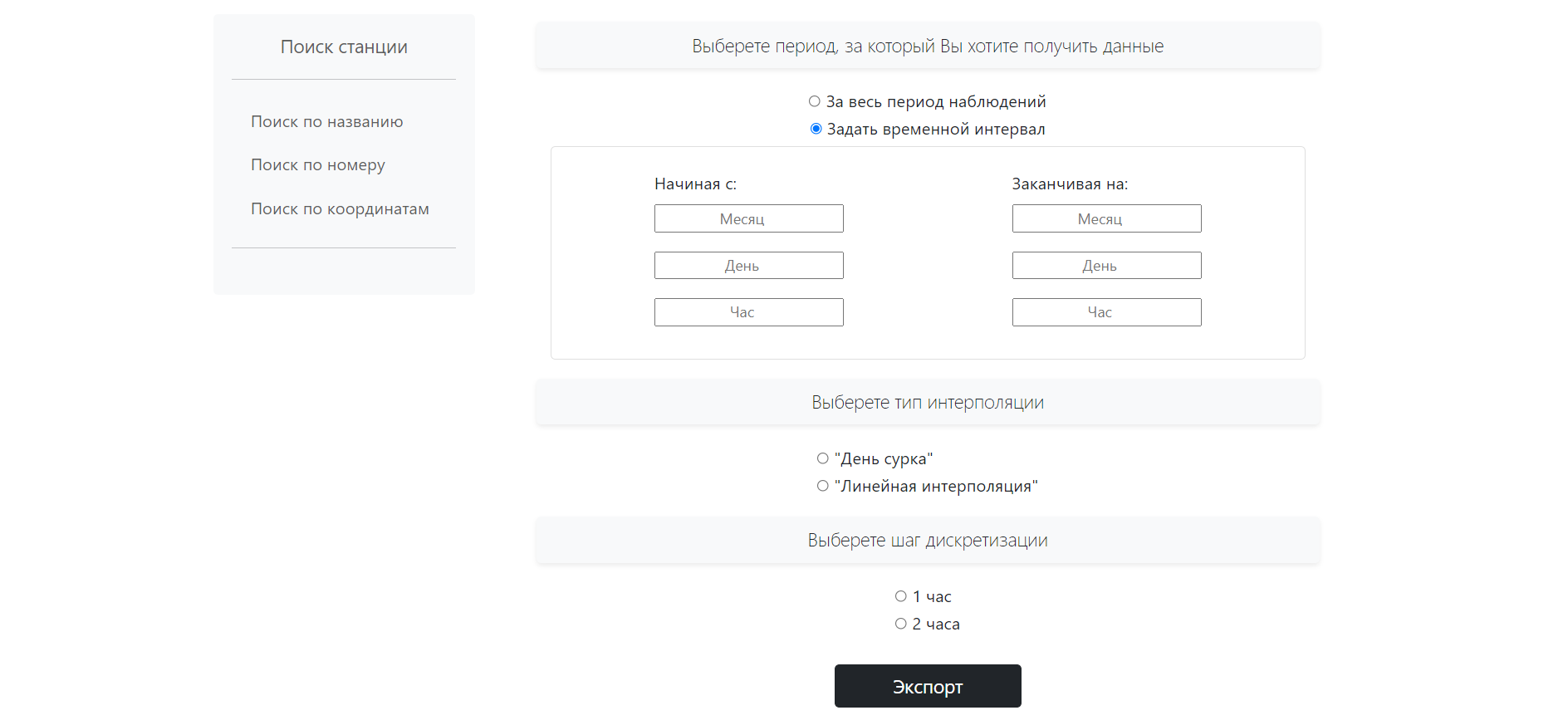
При переходе по ссылке «Интерполировать данные» пользователь попадает на страницу для задания параметров интерполяции. Страница содержит три группы по две радиокнопки. Первая группа отвечает за задание периода. Есть два варианта: «За весь период наблюдений» и «Задать временной интервал». При активации радиокнопки «Задать временной интервал» появляется выпадающая область, которая содержит 6 полей, три из которых относятся к заданию начала периода (месяц, день и час), а три остальных для задания окончания периода (месяц, день и час).

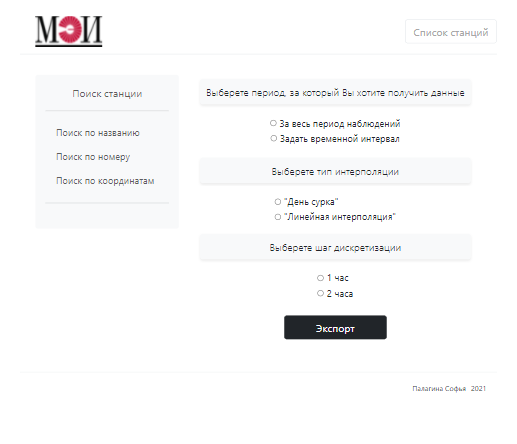
Вторая группа задает тип интерполяции и содержит два варианта: «День сурка» и «Линейная интерполяция».

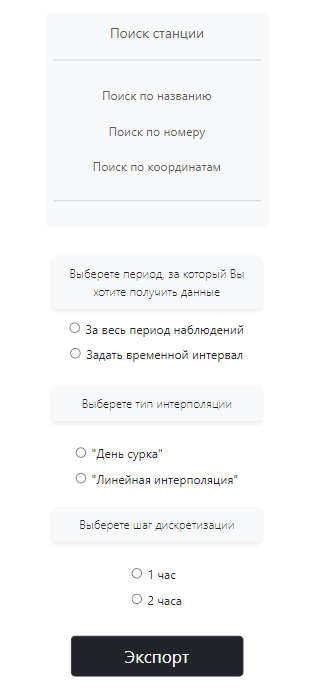
Третья группа задает шаг дискретизации, также содержит два варианта: «1 час» и «2 часа».

Пользователь может выбрать только один вариант из двух во всех группах. После выбора всех параметров необходимо нажать на кнопку «Экспорт», при нажатии на которую произойдет скачивание файла в формате csv. Файл будет содержать интерполированные данные в зависимости от заданных параметров. Вид пользовательского интерфейса представлен на Рис. .









Вид выходного файла представлен на Рис.

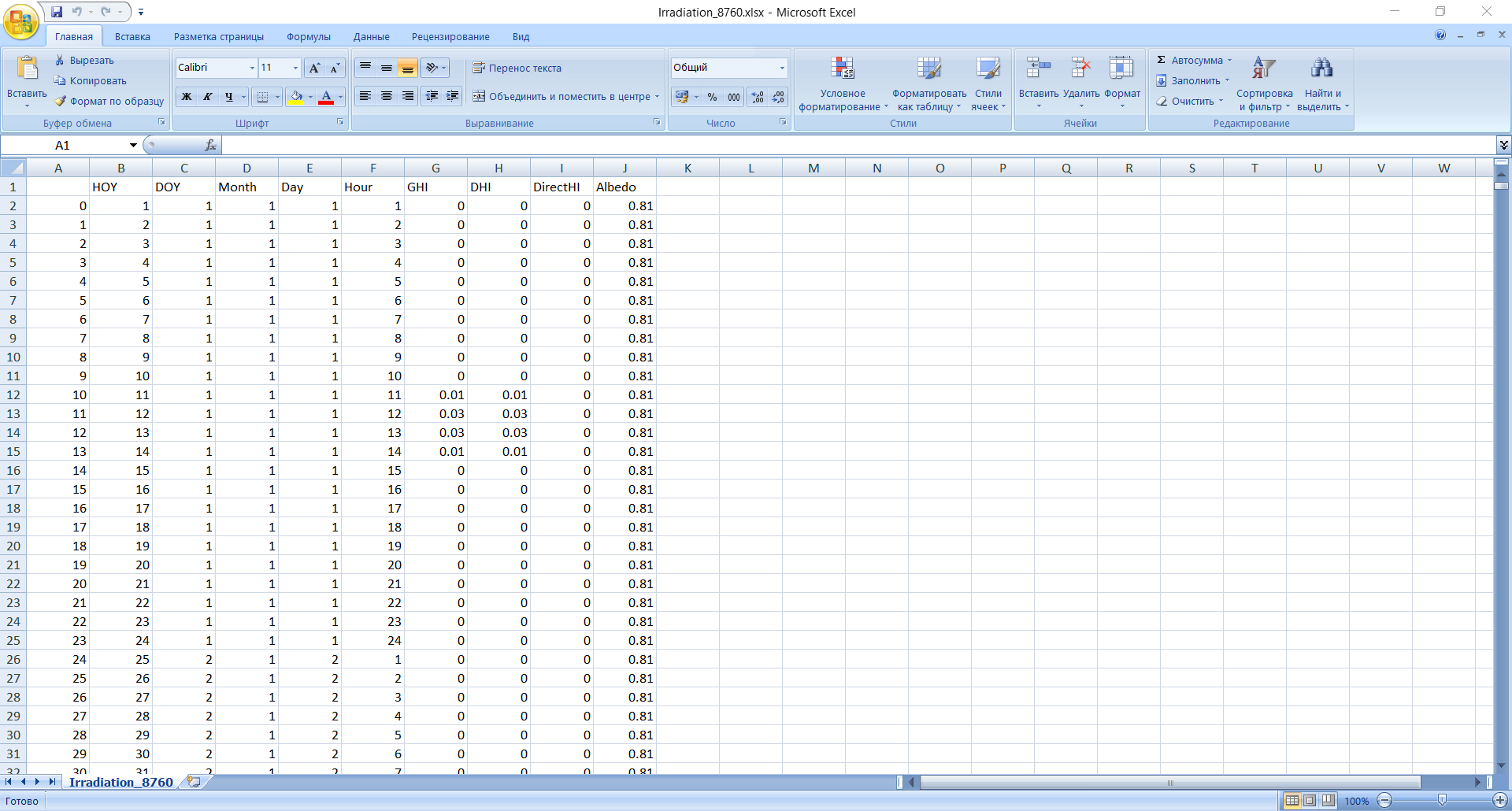


Рис. 7 Пример выходного файла csv

**Тестирование**

Тестирование приложения проводилось, в соответствии с межгосударственным стандартом ГОСТ 19.301-79 – «Программа и методика испытаний». Блок должен содержать следующие разделы:

1) Объект испытаний;

Объект тестирования – веб-приложение для поиска и обработки данных по актинометричесим станциям, которое было разработано в ходе выполнения данной выпускной квалификационной работы.

2) Цель испытаний;

Цель проведения испытаний – проверка работоспособности разаработанного приложения, выявление ошибок функционирования, оценка корректности работы.

3) Средства испытаний;

Тестирование производилось на ноутбуке со следующими характеристиками:

**Табл.**

**Параметры средства испытаний**

|  |  |
| --- | --- |
| Параметр | Значение |
| Операционная система | Windows 10 |
| Разрадность системы |  |
| Процессор | Intel(R) Core(TM) i7-8565U CPU 1.80GHz, 1992 МГц, ядер: 4, логических процессоров: 8 |
| ОЗУ | 8ГБ DDR3 2133 МГц |
| Разрешение экрана | 19201080 |

4) Методы испытаний;

Тесты нацелены на проверку работоспособности веб-приложения в случае ввода некорректных данных. Метод испытаний - функциональное тестирование по входным и выходным данным. Тестирование проводится в нормальных и исключительных условиях.

Все тесты были пройдены успешно, ошибок не выявлено.

**Тестирование алгоритма экспорта**

В процессе разработки алгоритма, были учтены все основные исключения.

Фраймворк Django имеет достаточно мощные механизмы валидации данных. То есть на клиентской части приложения будет автоматически идти проверка данных на корректность, прежде чем они отправятся на сервер [3].

Для полей *Period, Type, Hour* указан атрибут required, поэтому в случае, когда пользователь не выберет один из предложенных вариантов, фреймворк выдаст предупреждение. Также атрибут required имеет поле Месяц.

Для полей День и Час прописаны обработчики исключений. В случае, если пользователь не укажет День или Час, просто берутся значения по умолчанию.

Установлены максимальные и минимальные значения вводимых данных.

* для месяца: max=12, min=1;
* для дня: max=31, min=1;
* для часа: max=24, min1;

Поля Месяц, День и Час принимают только числовые значения, поэтому пользователь просто не сможет ввести символ или букву.

Помимо этого стоит учесть, что данные мы имеем за один год наблюдений, поэтому задавать период мы можем только в порядке возрастания. То есть мы не можем задать начало периода – 5-ый месяц, а конец периода – 2-ой месяц. Это касается и поля День, при условии, что месяцы начала и конца периода одинаковые (пользователь запросил данные за один месяц). Если указаны одинаковые Месяц и День, то час начала периода должен быть больше часа конца периода.

В случае некорректно заданного периода, генерируется исключение, файл не будет сформирован и данные не экспортируются, а пользователю придет сообщение об ошибке.

1. Пользователь ввел некорректный номер месяца, дня или часа (Рис. 10)

Рассмотрим данный случай на примере некорректно заданного месяца.

Входные данные: Месяц = 45

Получаем предупреждение, что значение не может быть больше 12.

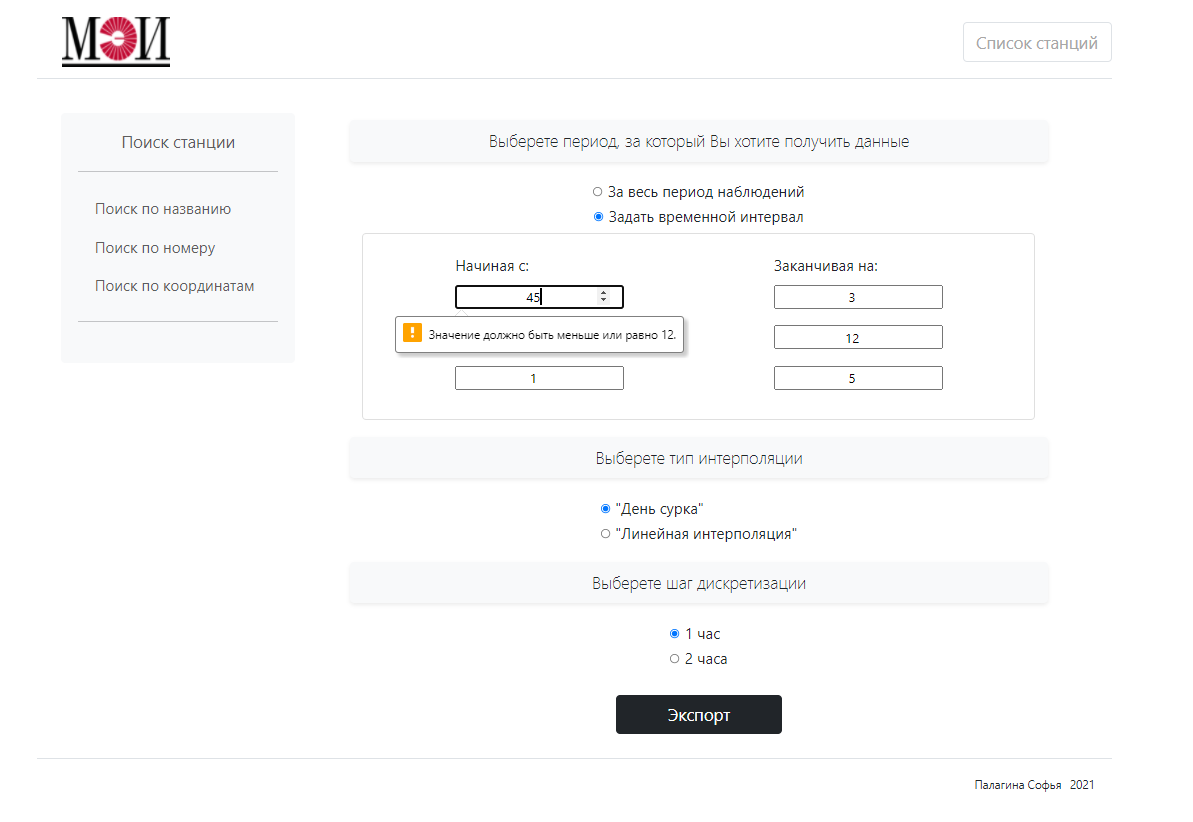


Рис. 10 Пример ввода некорректного числа в поле Месяц

2. Пользователь не выбрал один из вариантов

Рассмотрим случай, что пользователь не задал период, то есть оставил обе радиокнопки в группе задания периода неактивными (Рис. 11).

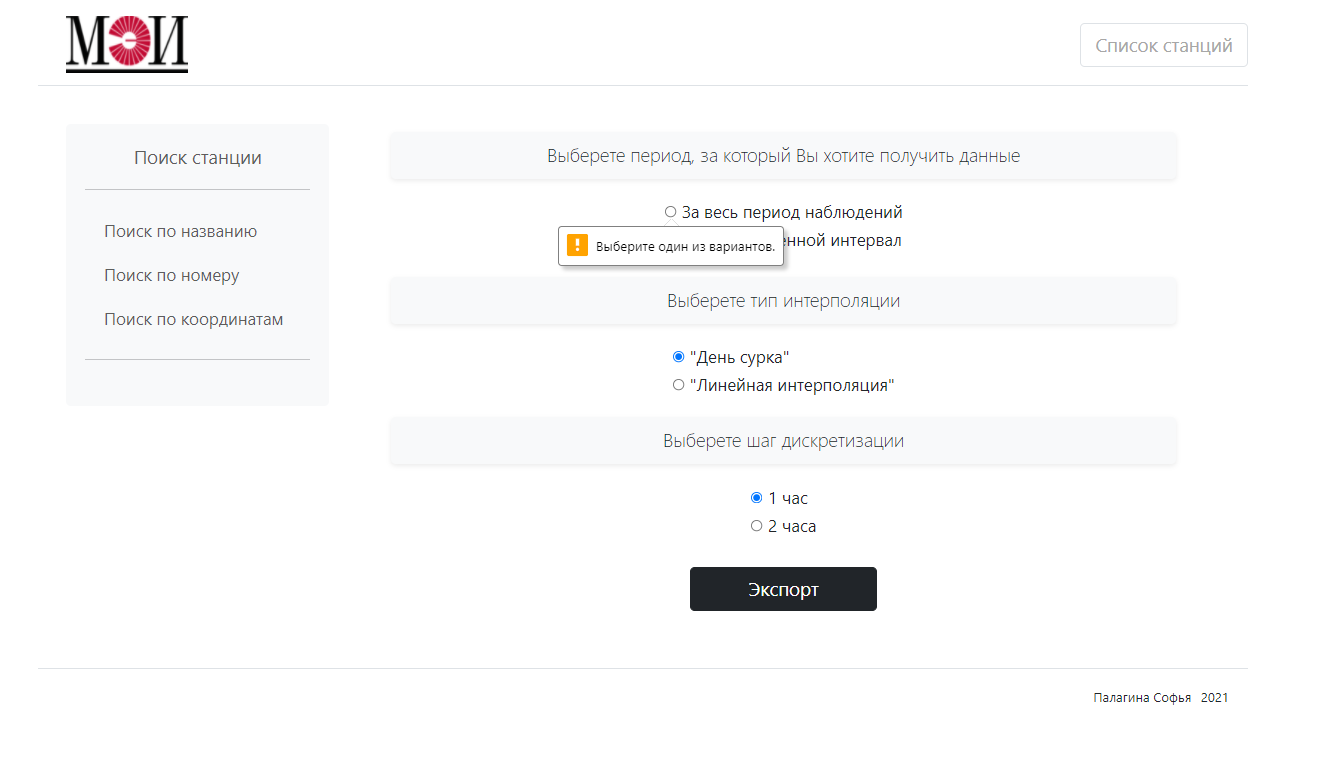


Рис. 11 Пример отсутствия выбора

Получаем предупреждение, что нужно выбрать один из предложенных вариантов.

3. Пользователь оставил обязательное поле незаполненным

В случае незаполненного обязательного поля - Месяц получаем предупреждение, что нужно поле заполнить (Рис. 12)

.

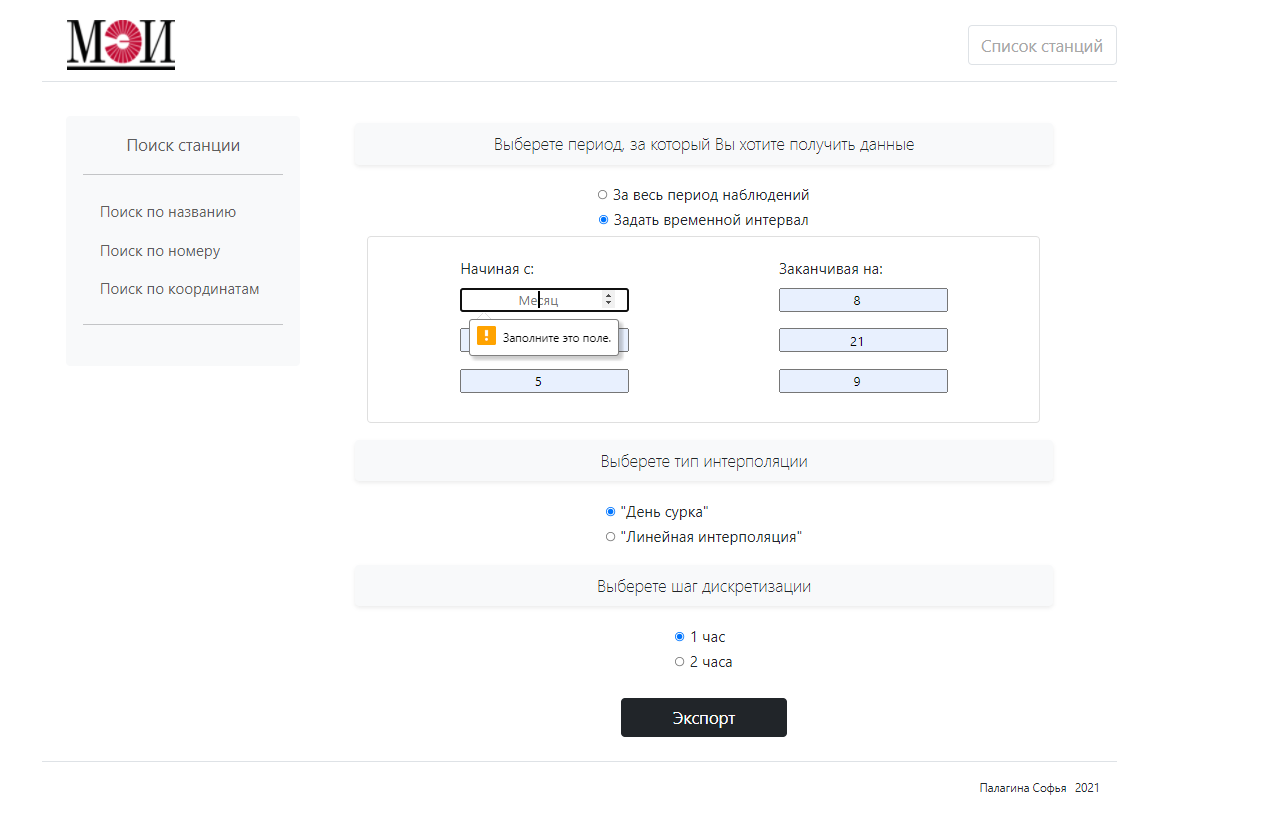


Рис. 12 Пример незаполненного обязательного поля

4. Пользователь неверно задал временной интервал (Рис. 13)

Входные данные:

Начало периода:

Месяц = 9

День = 8

Час = -

Завершение периода:

Месяц = 9

День =1

Час = -

Получается такая ситуация, что данные берутся за один месяц (за 9-ый), но при этом указывается, что начало периода это день – 8, а конец периода день – 1, то есть начало периода больше значения завершения периода.

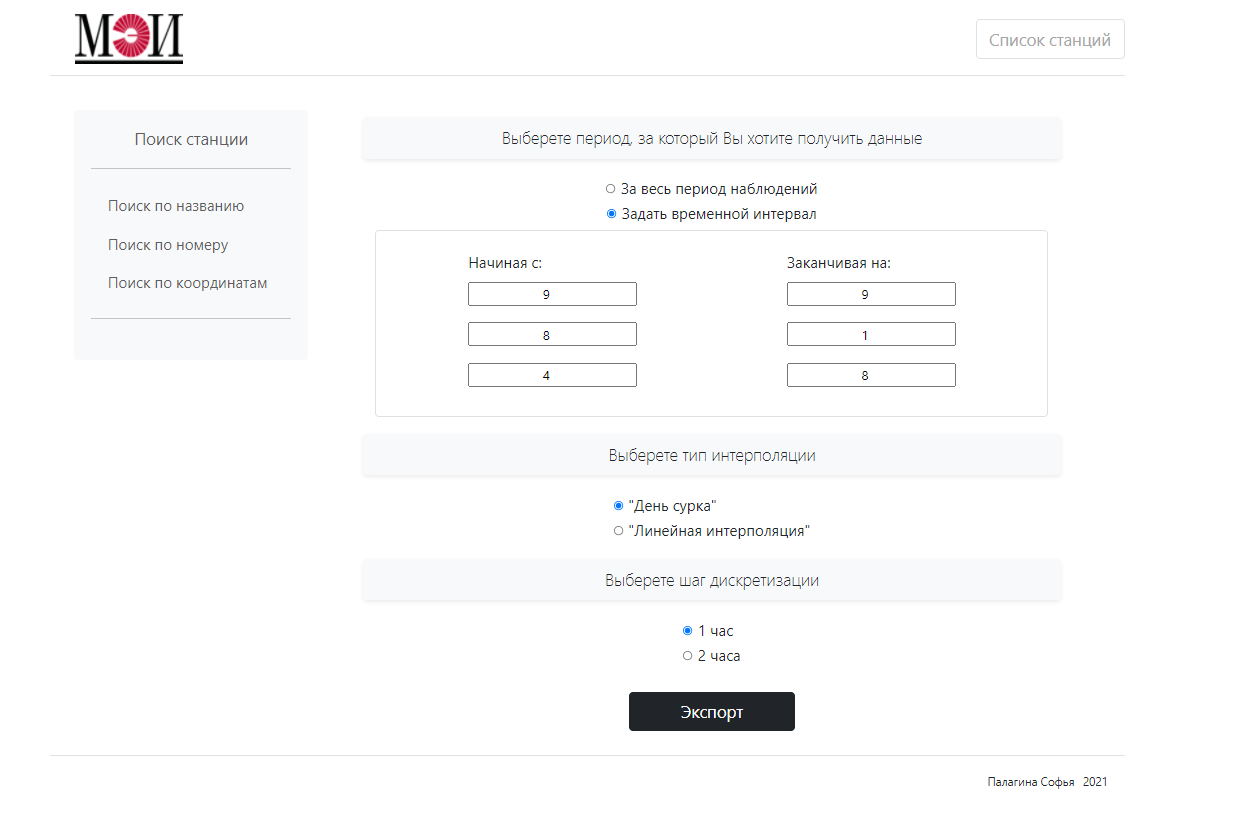


Рис. 13 Пример некорректного интервала

При попытке указать некорректный интервал, получаем сообщение об ошибке (Рис. 14).

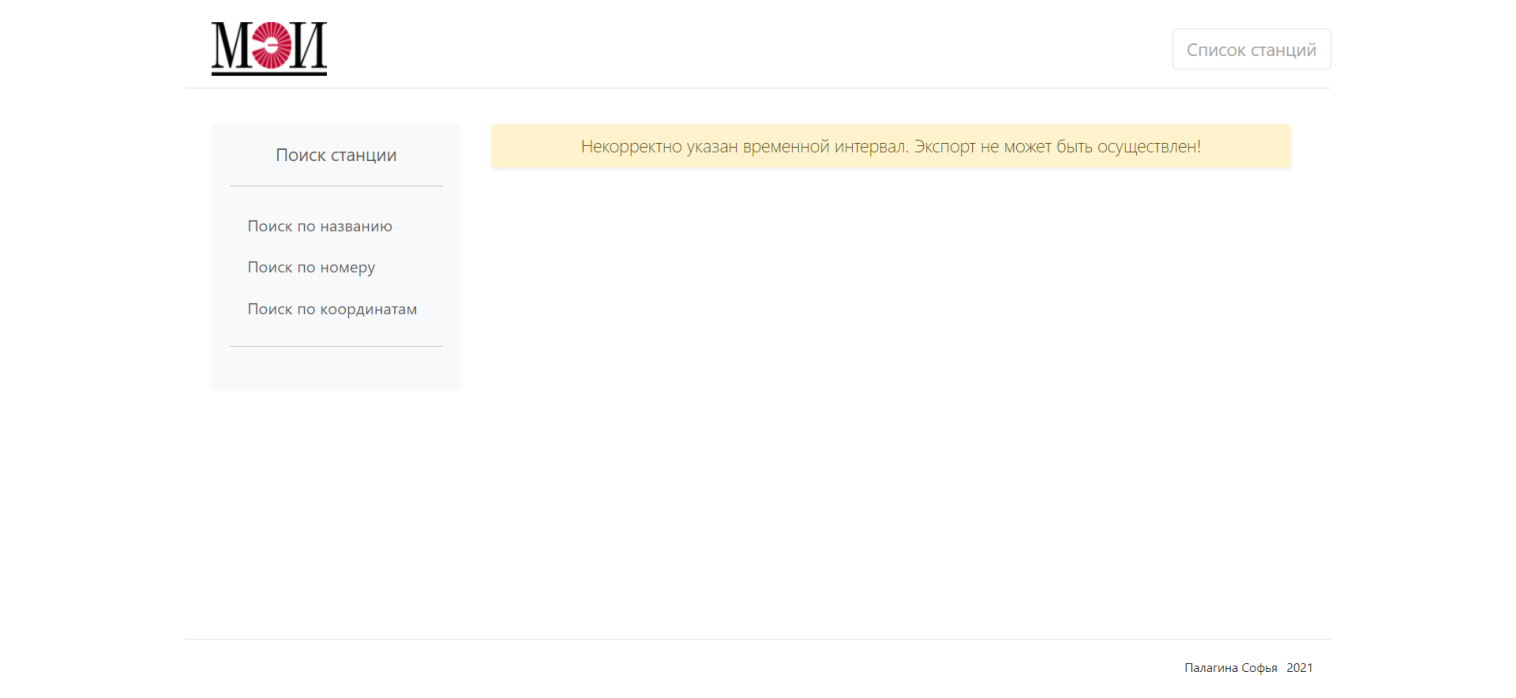


Рис. 14 Выводимое сообщение об ошибке

**Тестирование поиска**

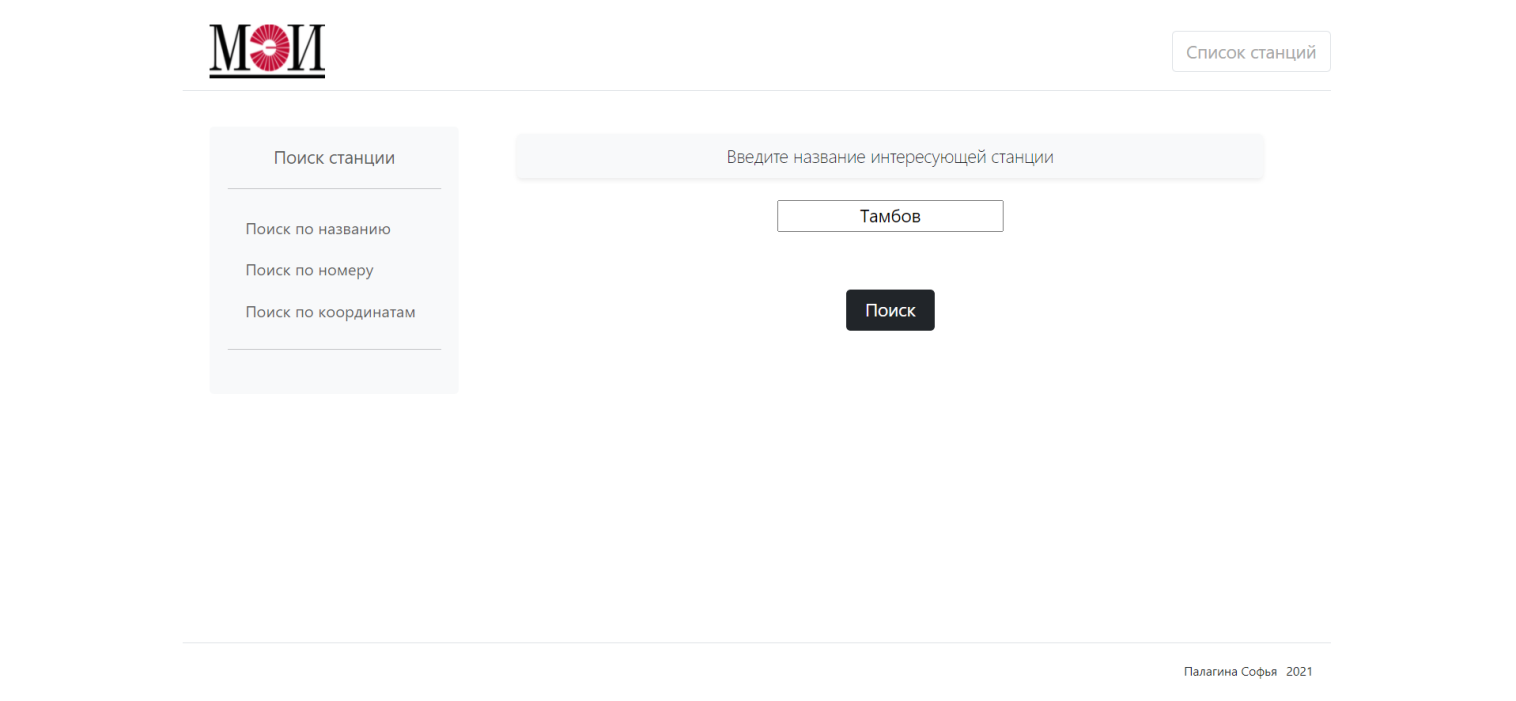
Рассматриваются случаи неверного ввода данных в поля поиска: «Поиск по названию», «Поиск по номеру» и «Поиск по координатам».

1. Пользователь ввел некорректное название станции

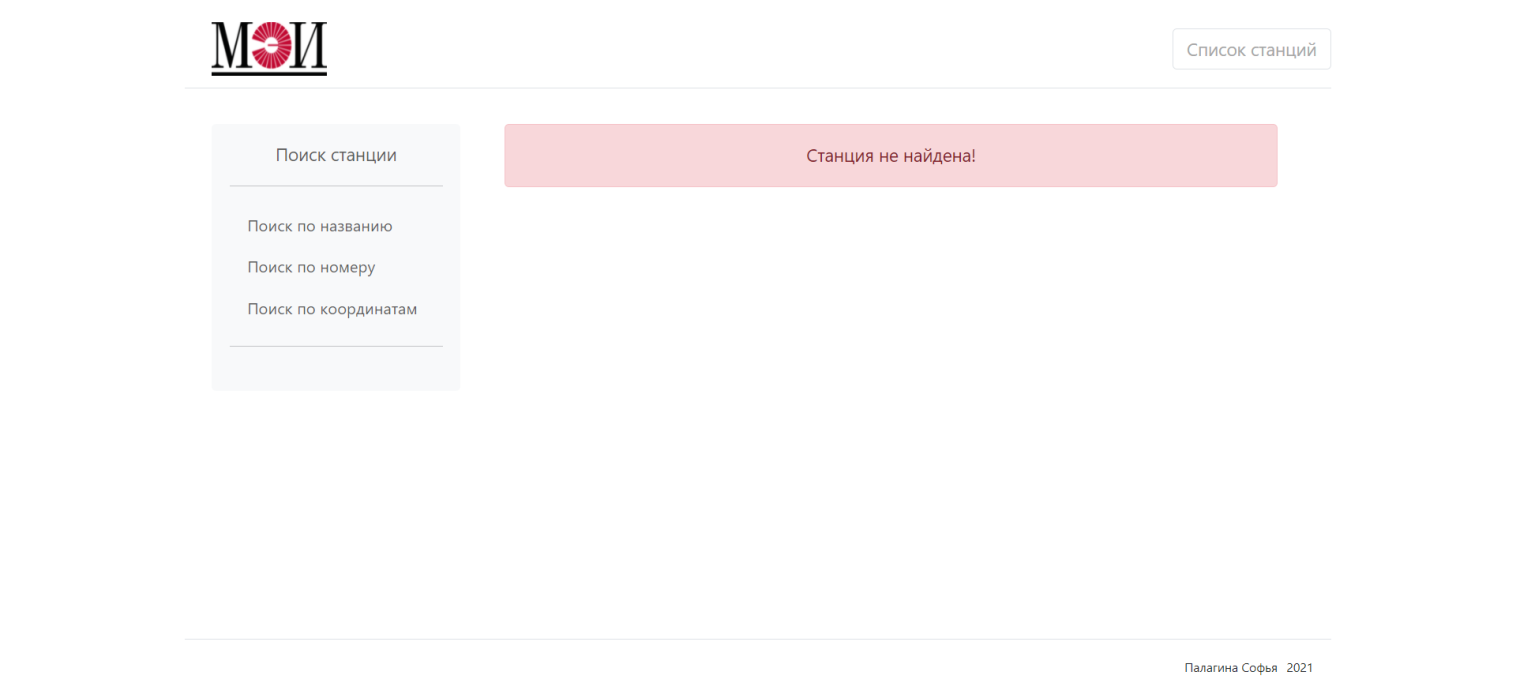
Случай, когда пользователь вводит в поле ввода название станции, информации по которой в базе данных не содержится (Рис. ).

Входные данные:

Название станции – Тамбов;

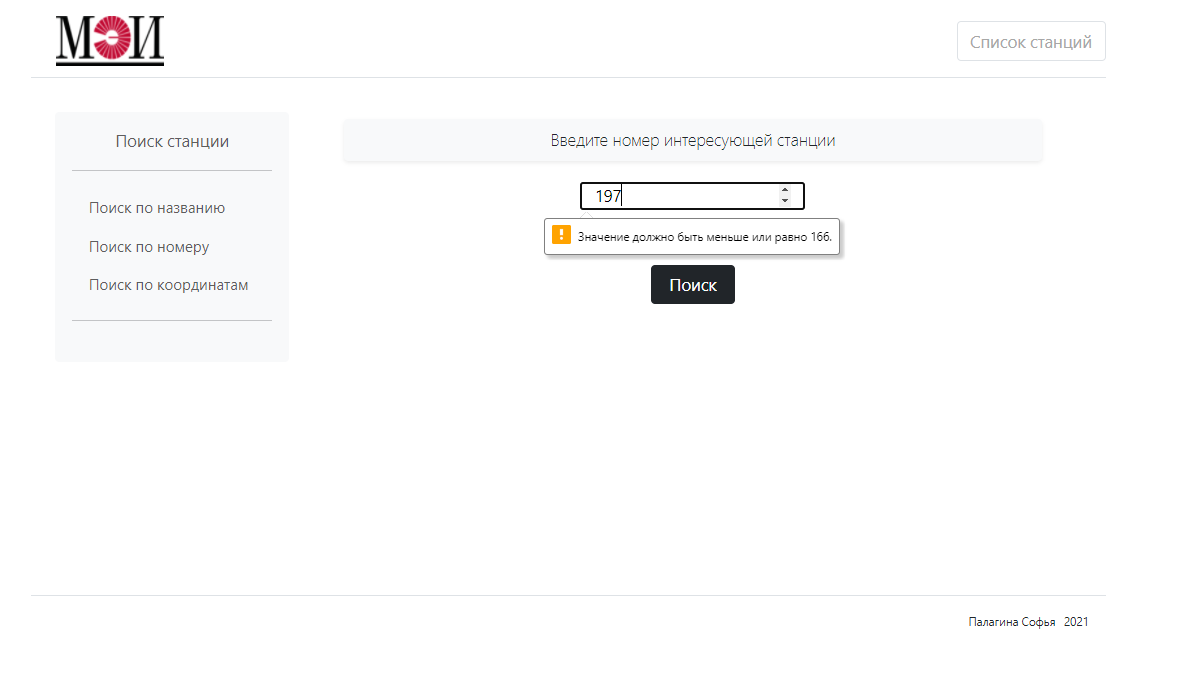


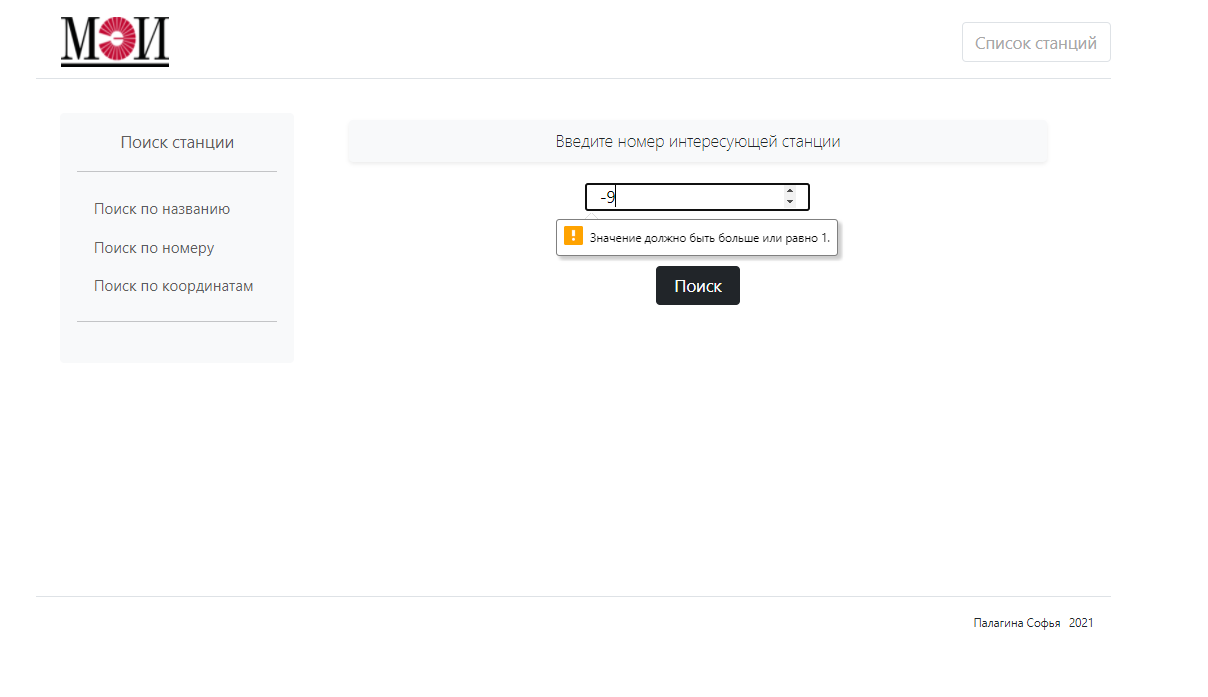
Станции с таким названием в базе данных не содержится, следовательно, как результат, получим предупреждение, что такой станции нет.



2. Пользователь указал некорректный номер станции

Если пользователь задст значение, равное номеру станции, вне интервала (1, 166), то появится предупреждение, что параметр некорректный (Рис. ).



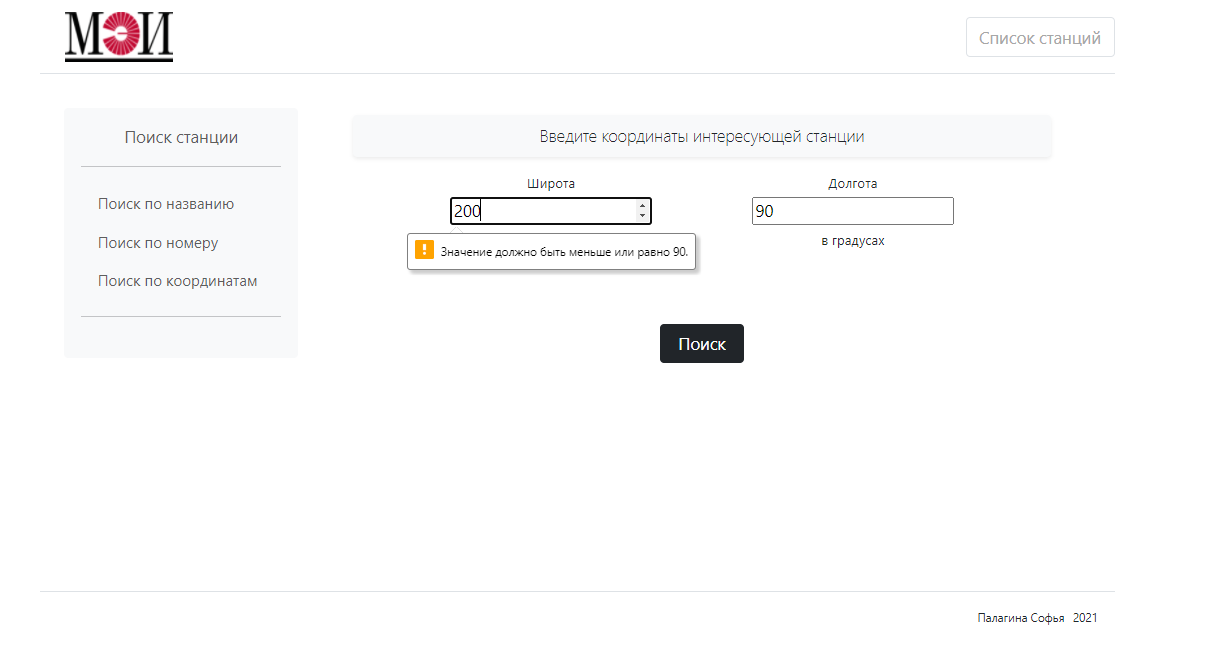


3. Пользователь указал некорректные географические координаты

Пользователь может ввести не совсем корректные значения для широты и долготы, в результате появится предупреждения о некорректности введенных данных (Рис. ).

Широта: 0 - 90

Долгота: 20 - 150



**Заключение**

В результате выполнения выпускной квалификационной работы было разработано действующее веб-приложение для поиска данных по актинометрическим станциям, с последуещей их обработкой. Разработанное приложение полностью удовлетворяет техническому заданию.

В ходе выполнения работы была разработана структура реляционной базы данных. База данных была приведена к третьей нормальной форме и построена таким образом, чтобы можно было с легкостью ее модифицировать, изменив или добавив новые компоненты. Было создано серверное приложение, позволяющее осуществлять быстрый и удобный поиск по базе данных. Разработаны алгоритмы интерполирования данных в зависимости от заданных параметров. Предусмотрен функционал для экспорта данных в формате csv. Разработанный пользовательский интерфейс интуитивно понятный, простой и отзывчивый к вводимым данным. Страницы приложения корректно отображаются в различных браузерах. Приложение было адаптировано под устройства с разным размером экрана.

Приложение прошло проверку на все основные исключения. Тестирование показало, что приложение работает корректно.

В будущем приложение можно будет с легкостью расширить и дополнить новым функционалом, если предъявляемые требования возрастут.

**Список использованной литературы**

1. Official site of Django framework django-import-export [Электронный ресурс]. – Режим доступа:

<https://django-import-export.readthedocs.io/en/latest/>

1. Official site of Django-documentation [Электронный ресурс]. – Режим доступа:

<https://docs.python.org/3/>

1. Official site of Django framework [Электронный ресурс]. – Режим доступа: <https://www.djangoproject.com/start/overview/>
2. <https://django.fun/docs/django/ru/3.1/>
3. <https://developer.mozilla.org/ru/docs/>
4. <https://geopy.readthedocs.io/en/stable/>
5. Шелдон. Р., Мойе Дж. MySQL: Базовый курс. : Пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 880 с.: ил.
6. С.А. Жданов, М.Л. Собоелва, А.С. Алфимова. Информационные системы: учебник студ. учреждений высш. образования – М.: ООО «Прометей», 2015. – 302 с
7. Фиайли К. SQL: Пер. с англ. – М.: ДМК Пресс. – 456 с.: ил. (Серия «Quick Start»).
8. <https://zametkinapolyah.ru/servera-i-protokoly/o-modeli-vzaimodejstviya-klient-server-prostymi-slovami-arxitektura-klient-server-s-primerami.html>
9. <https://testmatick.com/ru/osnovnye-ponyatiya-i-osobennosti-klient-servernoj-arhitektury/>
10. <https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/skeleton_website>
11. <https://bestprogrammer.ru/programmirovanie-i-razrabotka/kakoj-yazyk-luchshe-dlya-veb-razrabotki-php-ili-python>
12. <https://blog.skillfactory.ru/php-vs-python-chto-vybrat/>
13. <https://django.fun/tutorials/flask-vs-django-sravnenie-sozdaniya-rest-api/>
14. <https://pythonist.ru/django-vs-flask-chto-vybrat-dlya-svoego-proekta/>
15. <https://rtfm.co.ua/django-book-model-razrabotki-mtc-model-view-controller/>
16. <https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/>
17. https://www.vo-da.ru/book/klimat-09

**Приложение. Листинг серверной части программы**

**#Подпрограмма интерполяции по принципу «День сурка»**

**#Разработала Палагина С.А.**

**#Группа А-12-17**

#Перебор и сортировка входных данных, заполнение массивов для дальнейшей их обработки

for object in object\_list:  
 for field\_name, field\_value in object['fields'].items():

#Ищем по названию – ключу, значение записываем в массив  
 if field\_name == 'Month\_1': #заполнения массива данными за январь

a11.append(float(field\_value))  
 if field\_name == 'Month\_2': #заполнения массива данными за февраль

a12.append(float(field\_value))  
 if field\_name == 'Month\_3': #заполнения массива данными за март  
  
 a13.append(float(field\_value))  
 if field\_name == 'Month\_4': #заполнения массива данными за апрель  
   
 a14.append(float(field\_value))  
 if field\_name == 'Month\_5': #заполнения массива данными за май  
  
 a15.append(float(field\_value))  
 if field\_name == 'Month\_6': #заполнения массива данными за июнь  
  
 a16.append(float(field\_value))  
 if field\_name == 'Month\_7': #заполнения массива данными за июль  
  
 a17.append(float(field\_value))  
 if field\_name == 'Month\_8': #заполнения массива данными за август  
  
 a18.append(float(field\_value))  
 if field\_name == 'Month\_9': #заполнения массива данными за сентябрь  
  
 a19.append(float(field\_value))  
 if field\_name == 'Month\_10': #заполнения массива данными за октябрь  
  
 a110.append(float(field\_value))  
 if field\_name == 'Month\_11': #заполнения массива данными за ноябррь  
  
 a111.append(float(field\_value))  
 if field\_name == 'Month\_12': #заполнения массива данными за декабрь  
  
 a112.append(float(field\_value))

# Формирование выходной последовательности объеденением массивов

# «Склеиваем» все массивы в один при этом умножая их на коэффициенты, соответствующе числу дней в месяце

# Умножение массива на число эквивалентно его повторению указанное число раз  
a = a11 \* 31 + a12 \* 28 + a13 \* 31 + a14 \* 30 + a15 \* 31 + a16 \* 30 + a17 \* 31 + a18 \* 31 + a19 \* 30 + a110 \* 31 + a111 \* 30 + a112 \* 31

**#Подпрограмма интерполяции по принципу «Линейная интерполяция»**

**#Разработала Палагина С.А.**

**#Группа А-12-17**

# Заолнение выходного массива для промежутка между 15-м января и 15-м февраля

for i in range(24): #проводим данную операцию для каждого часа   
 raz = a12[i] - a11[i] #находим разницу между соседними элементами   
 delt = float(abs(raz) / 31) #находим шаг дискретизации, знаменатель зависит от числа дней между двумя точками  
 nach = a11[i] #запоминаем первую точку, относительно которой будем двигаться

#сравниваем разность с нулем   
 if raz < 0: #если разность меньше нуля, значит приход солнечной радиации уменьшается  
 for j in range(31):  
 nach = nach – delt #следовательно, вычитаем шаг  
 a21.append(float("{0:.3f}".format(nach))) #полученнумизначениями заполняем выходной массив  
 elif raz > 0: #если разность больше нуля, значит приход солнечной радиации увеличивается  
 for j in range(31):  
 nach = nach + delt #следовательно, прибавляем шаг  
 a21.append(float("{0:.3f}".format(nach)))  
 elif raz == 0: #если разность равна нулю, значит приход солнечной радиации не меняется  
 for j in range(31):  
 nach = nach + delt #просто прибавляем шаг  
 a21.append(float("{0:.3f}".format(nach)))

# процессы аналогичны  
# Заолнение выходного массива для промежутка между 15-м февраля и 15-м марта

for i in range(24):  
 raz = a13[i] - a12[i]  
 delt = float(abs(raz) / 28)  
 nach = a12[i]  
 if raz < 0:  
 for j in range(28):  
 nach = nach - delt  
 a22.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(28):  
 nach = nach + delt  
 a22.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(28):  
 nach = nach + delt  
 a22.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м марта и 15-м апреляfor i in range(24):  
 raz = a14[i] - a13[i]  
 delt = float(abs(raz) / 31)  
 nach = a13[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a23.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a23.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a23.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м апреля и 15-м маяfor i in range(24):  
 raz = a15[i] - a14[i]  
 delt = float(abs(raz) / 30)  
 nach = a14[i]  
 if raz < 0:  
 for j in range(30):  
 nach = nach - delt  
 a24.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(30):  
 nach = nach + delt  
 a24.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(30):  
 nach = nach + delt  
 a24.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м мая и 15-м июняfor i in range(24):  
 raz = a16[i] - a15[i]  
 delt = float(abs(raz) / 31)  
 nach = a15[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a25.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a25.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a25.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м июня и 15-м июляfor i in range(24):  
 raz = a17[i] - a16[i]  
 delt = float(abs(raz) / 30)  
 nach = a16[i]  
 if raz < 0:  
 for j in range(30):  
 nach = nach - delt  
 a26.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(30):  
 nach = nach + delt  
 a26.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(30):  
 nach = nach + delt  
 a26.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м июля и 15-м августаfor i in range(24):  
 raz = a18[i] - a17[i]  
 delt = float(abs(raz) / 31)  
 nach = a17[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a27.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a27.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a27.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м августа и 15-м сентябряfor i in range(24):  
 raz = a19[i] - a18[i]  
 delt = float(abs(raz) / 31)  
 nach = a18[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a28.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a28.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a28.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м сентября и 15-м октябряfor i in range(24):  
 raz = a110[i] - a19[i]  
 delt = float(abs(raz) / 30)  
 nach = a19[i]  
 if raz < 0:  
 for j in range(30):  
 nach = nach - delt  
 a29.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(30):  
 nach = nach + delt  
 a29.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(30):  
 nach = nach + delt  
 a29.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м октября и 15-м ноября

for i in range(24):  
 raz = a111[i] - a110[i]  
 delt = float(abs(raz) / 31)  
 nach = a110[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a210.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a210.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a210.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м ноября и 15-м декабряfor i in range(24):  
 raz = a112[i] - a111[i]  
 delt = float(abs(raz) / 30)  
 nach = a111[i]  
 if raz < 0:  
 for j in range(30):  
 nach = nach - delt  
 a211.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(30):  
 nach = nach + delt  
 a211.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(30):  
 nach = nach + delt  
 a211.append(float("{0:.3f}".format(nach)))  
*#* Заолнение выходного массива для промежутка между 15-м декабря и 15-м январяfor i in range(24):  
 raz = a11[i] - a112[i]  
 delt = float(abs(raz) / 31)  
 nach = a112[i]  
 if raz < 0:  
 for j in range(31):  
 nach = nach - delt  
 a212.append(float("{0:.3f}".format(nach)))  
 elif raz > 0:  
 for j in range(31):  
 nach = nach + delt  
 a212.append(float("{0:.3f}".format(nach)))  
 elif raz == 0:  
 for j in range(31):  
 nach = nach + delt  
 a212.append(float("{0:.3f}".format(nach)))

#разбиваем поледний массив, одна его часть, соответствующая промежутку с 1-го января по 15-е января, ставится в начало выходной последовательности, а вторая, соответствующая промежутку с 15-го декабря по 31-декабря, в конец

end = []  
for i in range(336):  
 end.append(a212[i])  
begin = []  
for i in range(336, 744):  
 begin.append(a212[i])

#формирование выходной последовательности посредством объединения всех массивов в нужном порядке

a = begin + a21 + a22 + a23 + a24 + a25 + a26 + a27 + a28 + a29 + a210 + a211 + end

**#Подпрограмма экспорта в формат csv**

**#Разработала Палагина С.А.**

**#Группа А-12-17**

def Export\_csv(request):

#получение параметров от других представлений  
 res0 = request.session.get('res3')   
 res = request.session.get('res2')   
 coord = request.session.get('coord')   
 res1 = request.session.get('res2\_num')   
 res2 = request.session.get('res\_coord\_1')   
 res3 = request.session.get('res\_coord\_2')   
 res4 = request.session.get('res\_coord\_3')   
 Export = request.GET.get("export")  
  
 response = HttpResponse(content\_type='text/csv') #экспортируем в формате csv  
 writer = csv.writer(response)  
 writer.writerow(['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12'])

#если пользователь нажал на кнопку экспорт Альбедо  
 if Export == 'Albedo':  
 if res0:  
 response['Content-Disposition'] = 'attachment; filename = "Typical\_Albedo12.csv"' #экспортируем с названием "Typical\_Albedo12.csv"

obj = next(serializers.deserialize("json", res0)).object #сериализуем данные для вывода

writer.writerow([obj.Month\_1, obj.Month\_2, obj.Month\_3, obj.Month\_4, obj.Month\_5, obj.Month\_6, obj.Month\_7, obj.Month\_8, obj.Month\_9, obj.Month\_10, obj.Month\_11, obj.Month\_12])

#получение id  
 if res: #если пришло в параметре название станции  
 r = StationsNew.objects.get(NameStation=res)

#если пришло в параметре id станции  
 elif res1:   
 r = StationsNew.objects.get(id=res1)  
 elif coord == '1':  
 if res2:  
 r = StationsNew.objects.get(id=res2)  
 elif coord == '2':  
 if res3:  
 r = StationsNew.objects.get(id=res3)  
 elif coord == '3':  
 if res4:  
 r = StationsNew.objects.get(id=res4)  
  
#если пользователь выбрал экспорт среднемесячные суммы суммарной/прямой/диффузной солнечной радиации  
 if Export == 'Sum': #если суммарная  
 response['Content-Disposition'] = 'attachment; filename = "Typical\_GlobalHI24.csv"' #указываем название   
 stat = hour\_sum\_sum\_srNew.objects.filter(IdStations=r.id) #получаем соответствующие данные

if Export == 'Straight': #если прямая  
 response['Content-Disposition'] = 'attachment; filename = "Typical\_DirectHI24.csv"'#указываем название  
 stat = hour\_sum\_straight\_srNew.objects.filter(IdStations=r.id)  
  
 if Export == 'Diff': #если диффузная  
 response['Content-Disposition'] = 'attachment; filename = "Typical\_DiffuseHI24.csv"'#указываем название  
 stat = hour\_sum\_diff\_srNew.objects.filter(IdStations=r.id)  
  
 for row in stat:  
 writer.writerow([row.IdHour, row.Month\_1, row.Month\_2, row.Month\_3, row.Month\_4, row.Month\_5, row.Month\_6, row.Month\_7, row.Month\_8, row.Month\_9, row.Month\_10, row.Month\_11, row.Month\_12])  
  
 return response

**#Подпрограмма экспорта в формате csv интерполированных данных**

**#Разработала Палагина С.А.**

**#Группа А-12-17**

def Ex(request):

#получаем данные от других представлений  
 res = request.session.get('res2')  
 coord = request.session.get('coord')  
 res1 = request.session.get('res2\_num')  
 error = ''  
#получение id в зависимости от прешедших параметров  
 if res:  
 r = StationsNew.objects.get(NameStation=res)  
 if res1:  
 r = StationsNew.objects.get(id=res1)  
 if coord == '1':  
 res2 = request.session.get('res\_coord\_1')  
 r = StationsNew.objects.get(id=res2)  
 if coord == '2':  
 res3 = request.session.get('res\_coord\_2')  
 r = StationsNew.objects.get(id=res3)  
 if coord == '3':  
 res4 = request.session.get('res\_coord\_3')  
 r = StationsNew.objects.get(id=res4)

#получаем исходные данные по id от тредуемых моделей  
 stat = AlbedoNew.objects.filter(id=r.id)  
 stat1 = hour\_sum\_sum\_srNew.objects.filter(IdStations=r.id)  
 stat2 = hour\_sum\_straight\_srNew.objects.filter(IdStations=r.id)  
 stat3 = hour\_sum\_diff\_srNew.objects.filter(IdStations=r.id)

#подготовка второстепенных массивов для вывода выходного файла

#столбец Hour  
 hour = []  
 for i in range(365):  
 for j in range(1, 25):  
 hour.append(j)

#столбец день  
 day = []  
 for i in range(1, 13):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 for j in range(1, 32):  
 for k in range(24):  
 day.append(j)  
 elif i in [4, 6, 9, 11]:  
 for j in range(1, 31):  
 for k in range(24):  
 day.append(j)  
 elif i == 2:  
 for j in range(1, 29):  
 for k in range(24):  
 day.append(j)  
#столбец месяц   
 month = []  
 for i in range(1, 13):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 for j in range(31):  
 for k in range(24):  
 month.append(i)  
 elif i in [4, 6, 9, 11]:  
 for j in range(30):  
 for k in range(24):  
 month.append(i)  
 elif i == 2:  
 for j in range(28):  
 for k in range(24):  
 month.append(i)

#столбец день в году  
 DOY = []  
 for i in range(1, 366):  
 for j in range(24):  
 DOY.append(i)

#столбец час в году  
 HOY = []  
 for i in range(1, 8761):  
 HOY.append(i)  
  
 ind = []  
 for i in range(4380):  
 ind.append(i)  
  
 per = []  
 for i in range(8760):  
 per.append(i)  
 del per[::2]  
  
#сериализуем входные данные для удобного перебора  
 object\_list = serializers.serialize("python", stat)  
 object\_list1 = serializers.serialize("python", stat1)  
 object\_list2 = serializers.serialize("python", stat2)  
 object\_list3 = serializers.serialize("python", stat3)  
  
#получаем входные параметры от пользовательской формы  
 Period = request.GET.get('Period') #период, за который берем данные  
 Type = request.GET.get('Type') #тип интерполяции, который будем применять  
 Hour = request.GET.get('Hour') #шаг дискретизации, с которым будем идти  
  
 *# Тип интерполяции - "День сурка"* if Type == 'Type\_surok':

*# заполнение массивов  
 # процесс интерполирования*

*# получаем выходные массивы заполненные интерполированными данными*  
  
 *# Данные за весь период* if Period == 'All\_period':

*# Шаг дискретизации 1 час* if Hour == 'Hour\_1':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])

for i in range(8760): #берем все элементы массивов  
 writer.writerow([i, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
  
 *# Шаг дискретизации 2 часа* if Hour == 'Hour\_2':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
  
 j = 0  
 for i in per: #берем каждый второй элемент из массивов   
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j+1

*# Данные за указанный интервал* if Period == 'Part\_Period':

#высчитываем индекс, с которого необходимо начать брать данные  
 Mes\_begin = request.GET.get('Mes\_begin')  
 Day\_begin = request.GET.get('Day\_begin')

#обработка случая, когда не указан день или часа  
 if Day\_begin == '':  
 Day\_begin = 1  
 Hour\_begin = request.GET.get('Hour\_begin')  
 if Hour\_begin == '':  
 Hour\_begin = 1  
  
 #высчитываем индекс, на котором необходимо завершить брать данные  
 Mes\_end = request.GET.get('Mes\_end')  
 Day\_end = request.GET.get('Day\_end')

#обработка случая, когда не указан день или часа

#в зависимости от месяца берем последний день  
  
 if int(Mes\_end) in [1, 3, 5, 7, 8, 10, 12]:   
 Day\_end = 31  
 if int(Mes\_end) == 2:  
 Day\_end = 28  
 if int(Mes\_end) in [4, 6, 9, 11]:  
 Day\_end = 30

Hour\_end = request.GET.get('Hour\_end')  
 if Hour\_end == '':  
 Hour\_end = 24

#проверка на корректность указания интервала   
if int(Mes\_begin) > int(Mes\_end):  
 return render(request, 'MainApp/Error.html')  
  
if int(Mes\_begin) == int(Mes\_end):  
 if int(Day\_begin) > int(Day\_end):  
 return render(request, 'MainApp/Error.html')  
  
if int(Mes\_begin) == int(Mes\_end):  
 if int(Day\_begin) == int(Day\_end):  
 if int(Hour\_begin) > int(Hour\_end):  
 return render(request, 'MainApp/Error.html')

sum\_begin = 0  
 sum\_end = 0  
  
 for i in range(1, int(Mes\_begin)):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 sum\_begin = sum\_begin + 744  
 if i == 2:  
 sum\_begin = sum\_begin + 672  
 if i in [4, 6, 9, 11]:  
 sum\_begin = sum\_begin + 720  
  
 sum\_begin = sum\_begin + int((int(Day\_begin)-1)\*24)  
 sum\_begin = sum\_begin + int(Hour\_begin)  
  
 for i in range(1, int(Mes\_end)):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 sum\_end = sum\_end + 744  
 if i == 2:  
 sum\_end = sum\_end + 672  
 if i in [4, 6, 9, 11]:  
 sum\_end = sum\_end + 720  
  
 sum\_end = sum\_end + int((int(Day\_end)-1)\*24)  
 sum\_end = sum\_end + int(Hour\_end)  
  
 if sum\_begin < sum\_end:

*# Шаг дискретизации 1 час* if Hour == 'Hour\_1':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
 j = 0  
 for i in range(sum\_begin-1, sum\_end):  
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j+1  
  
 *# Шаг дискретизации 2 часа* if Hour == 'Hour\_2':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
 j = 0  
 for i in per:  
 if i in range(sum\_begin-1, sum\_end+1):  
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j+1  
 else:  
 error = 'e'  
 return render(request, 'MainApp/Export.html', context={'error':error})  
 error = ''  
  
*# Тип интерполяции - линейная* if Type == 'Type\_lin':  
  
  
 *# заполнение массивов  
 # процесс интерполирования*

*# получаем выходные массивы заполненные интерполированными данными*

*# За весь период*  
 if Period == 'All\_period':

*# Шаг дискретизации 1 час* if Hour == 'Hour\_1':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
  
 for i in range(8760):  
 writer.writerow([i, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
  
 *# Шаг дискретизации 2 часа* if Hour == 'Hour\_2':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
  
 j = 0  
 for i in per:  
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j + 1  
  
 *# Данные за указанный интервал* if Period == 'Part\_Period':

#получение индекса начала

Mes\_begin = request.GET.get('Mes\_begin')  
Day\_begin = request.GET.get('Day\_begin')

#обработка случая, когда не указан день или час  
if Day\_begin == '':  
 Day\_begin = 1  
Hour\_begin = request.GET.get('Hour\_begin')  
if Hour\_begin == '':  
 Hour\_begin = 1  
  
#получение индекса завершения  
Mes\_end = request.GET.get('Mes\_end')  
Day\_end = request.GET.get('Day\_end')

#обработка случая, когда не указан день или час  
if int(Mes\_end) in [1, 3, 5, 7, 8, 10, 12]:  
 Day\_end = 31  
if int(Mes\_end) == 2:  
 Day\_end = 28  
if int(Mes\_end) in [4, 6, 9, 11]:  
 Day\_end = 30  
Hour\_end = request.GET.get('Hour\_end')  
if Hour\_end == '':  
 Hour\_end = 24

#проверка на корректность указания интервала   
if int(Mes\_begin) > int(Mes\_end):  
 return render(request, 'MainApp/Error.html')  
  
if int(Mes\_begin) == int(Mes\_end):  
 if int(Day\_begin) > int(Day\_end):  
 return render(request, 'MainApp/Error.html')  
  
if int(Mes\_begin) == int(Mes\_end):  
 if int(Day\_begin) == int(Day\_end):  
 if int(Hour\_begin) > int(Hour\_end):  
 return render(request, 'MainApp/Error.html')

sum\_begin = 0  
 sum\_end = 0  
  
 for i in range(1, int(Mes\_begin)):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 sum\_begin = sum\_begin + 744  
 if i == 2:  
 sum\_begin = sum\_begin + 672  
 if i in [4, 6, 9, 11]:  
 sum\_begin = sum\_begin + 720  
  
 sum\_begin = sum\_begin + int((int(Day\_begin) - 1) \* 24)  
 sum\_begin = sum\_begin + int(Hour\_begin)  
  
 for i in range(1, int(Mes\_end)):  
 if i in [1, 3, 5, 7, 8, 10, 12]:  
 sum\_end = sum\_end + 744  
 if i == 2:  
 sum\_end = sum\_end + 672  
 if i in [4, 6, 9, 11]:  
 sum\_end = sum\_end + 720  
  
 sum\_end = sum\_end + int((int(Day\_end) - 1) \* 24)  
 sum\_end = sum\_end + int(Hour\_end)  
  
 *# Шаг дискретизации 1 час* if Hour == 'Hour\_1':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
 j = 0  
 for i in range(sum\_begin - 1, sum\_end):  
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j + 1  
  
 *# Шаг дискретизации 2 часа* if Hour == 'Hour\_2':  
  
 response = HttpResponse(content\_type='text/csv')  
 writer = csv.writer(response)  
 response['Content-Disposition'] = 'attachment; filename = "Irradiation\_8760.csv"'  
  
 writer.writerow([' ', 'HOY', 'DOY', 'Month', 'Day', 'Hour', 'GHI', 'DHI', 'DirectHI', 'Albedo'])  
 j = 0  
 for i in per:  
 if i in range(sum\_begin - 1, sum\_end + 1):  
 writer.writerow([j, HOY[i], DOY[i], month[i], day[i], hour[i], a1[i], a3[i], a2[i], a[i]])  
 j = j + 1  
 return response