МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
профессор
_____ Л.Б. Соколинский
«___»_____ 2024г.

Разработка настольного приложения для анализа заболоченности озера

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-538.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Макаровских Т.А.

Автор работы,
студент группы КЭ-402
_____ А.К. Дымов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой СП

_____ Л.Б. Соколинский

29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-402
Дымову Артему Константиновичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка настольного приложения для анализа заболоченности озера.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Pytiff. [Электронный ресурс] URL: https://pytiff.readthedocs.io (дата обращения: 09.02.2024 г.).

3.2. Tkinter. [Электронный ресурс] URL: https://habr.com/ru/articles/133337 (дата обращения: 09.02.2024 г.).

3.3. EarthExplorer. [Электронный ресурс] URL: https://www.usgs.gov/tools (дата обращения: 09.02.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Анализ существующих ГИС-систем.

4.2. Изучить методы работы с изображениями TIFF формата.

4.3. Изучить библиотеки для создания графического интерфейса на языке Python.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению А.К. Дымов

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

С природой и водными ресурсами связана критически важная роль в жизни общества. Вместе с тем, увеличение загрязнений водных объектов поднимает вопрос необходимости эффективного контроля за состоянием водоемов, особенно в контексте заболоченности.

Существующие методы сбора и обработки данных о заболоченности водоемов часто неэффективны и трудоемки. Это может влиять на экосистемы водных объектов и становится проблемой для управления водными ресурсами.

В результате подчеркивается актуальность создания приложения для анализа заболоченности водоемов. Такое приложение должно предоставлять удобные инструменты для обработки и визуализации данных, что поможет экологам и специалистам водных ресурсов эффективнее решать проблемы, связанные с заболоченностью водных объектов.

Постановка задачи

Целью выпускной квалификационной работы является создание приложения для анализа заболоченности водоемов. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор аналогичных систем;
2) разработать структуру и функциональную часть приложения;
3) разработать Python приложение, обеспечивающее анализ данных о заболоченности водных объектов;
4) реализовать графический интерфейс для пользовательского взаимодействия;
5) провести тестирование приложения;
6) опубликовать приложение на GitHub.

Структура и содержание работы

Работа состоит из введения, трех глав, заключения и списка литературы. Объем работы составляет 32 страницы, объем списка литературы – 15 источников.

Первая глава посвящена анализу предметной области, обзору аналогичных проектов, а также рассмотрению основных библиотек, необходимых для выполнения поставленных задач. В ней приводится детальное описание существующих решений и их функциональных возможностей, достоинств и недостатков, что позволяет определить направление разработки и требования к создаваемой системе.

Вторая глава посвящена проектированию разрабатываемой системы. Выявляются основные требования к системе, функциональные и нефункциональные требования к приложению, а также разрабатывается диаграмма вариантов использования.

В третьей главе были описаны программные средства реализации системы, а также приведены детали реализации и листинги модулей приложения, связанных с созданием графического интерфейса и расчетом индекса NDVI, иллюстрирующие ключевые аспекты реализации системы.

Четвертая глава посвящена тестированию системы. В ней были описаны модульные тесты системы, а также результаты тестирования.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области и обзор аналогичных решений

Основная задача заключается в создании настольного приложения для анализа заболоченности озера на языке Python [9]. Приложение будет предоставлять возможность анализировать данные по заболачиванию озер, используя снимки со спутников, по которым можно определить индекс NDVI.

NDVI (Normalized Difference Vegetation Index) – нормализованный относительный индекс растительности – простой показатель количества фотосинтетической активной биомассы. Этот индекс часто используется для оценки состояния растительности и мониторинга процессов заболачивания водоемов.

Для реализации данного приложения будут использованы карты формата TIFF и shape-файлы, предоставленные УралГИСАгро. Эти данные содержат необходимую информацию о состоянии озер и окружающей их растительности, что позволит провести анализ заболоченности водоемов.

Приложение будет разработано на языке программирования Python, который обладает широким набором библиотек для работы с геопространственными данными и их визуализации. Пользователи смогут загружать спутниковые снимки и shape-файлы, а приложение будет проводить расчет индекса NDVI и отображать результаты анализа.

Обзор аналогичных решений представляет собой систематический анализ существующих инструментов, применяемых в области изучения заболоченности озер. Обзор поможет определить основные особенности, преимущества и недостатки, а также функциональность каждого из решений.

SentinelHub [14]

Одним из аналоговых приложений для анализа заболоченности является SentinelHub, которое предоставляет возможность анализировать данные по заболачиванию озер, и получать информацию о глубине, площади,

наличии растительности и других параметрах. Функционал приложения приведен на рисунке 1.

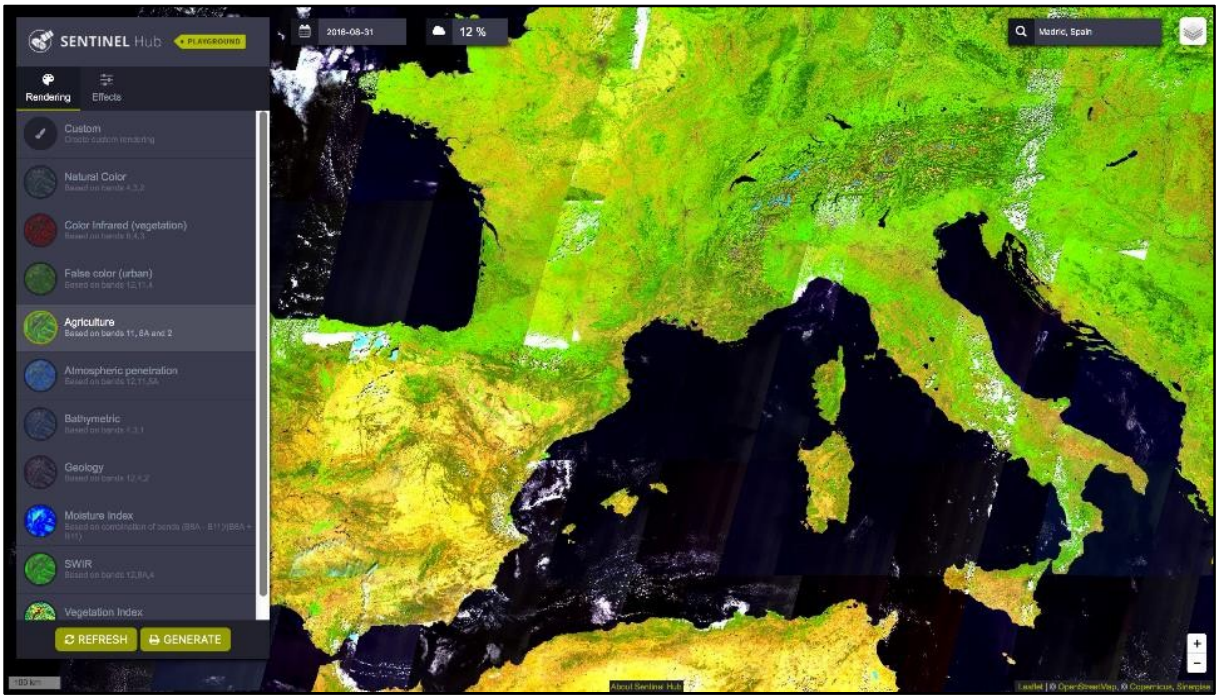

Рисунок 1 – Функционал SentinelHub

Данный ресурс уникален тем, что имеет большую выборку индексов, по которым можно осуществить анализ как водоемов, так и суши, а также имеет большой выбор снимков разных годов и с разными погодными условиями.

Платформа SentinelHub предоставляет широкий спектр инструментов и возможностей для комплексного анализа водных объектов.

Одним из ключевых инструментов является индекс водной зелени (WaterVegetation Index, WVI), который позволяет количественно оценить содержание фитопланктона в водоеме на основе данных, полученных от различных спутниковых систем, таких как Landsat и Sentinel-2.

SentinelHub также предоставляет доступ к данным об уровне воды в реках и озерах, которые могут быть использованы для мониторинга изменений водоемов.

Другим важным инструментом являются оптические изображения высокого разрешения, которые могут быть использованы для оценки каче-

ства воды, картирования водной растительности, а также для определения пространственных характеристик водных объектов.

Платформа SentinelHub позволяет применять алгоритмы автоматизированной классификации для определения типов водных объектов, таких как озера, реки и болота, что значительно упрощает процесс их инвентаризации и изучения.

QGIS [12]

Еще одним программным обеспечением, которое может быть использовано для анализа процессов заболачивания озер, является геоинформационная система с открытым исходным кодом QGIS. Данное приложение предоставляет широкий набор инструментов для обработки и анализа пространственных данных, позволяя оценивать состояние водных объектов на основе различных показателей. Интерфейс программы приведен на рисунке 2.

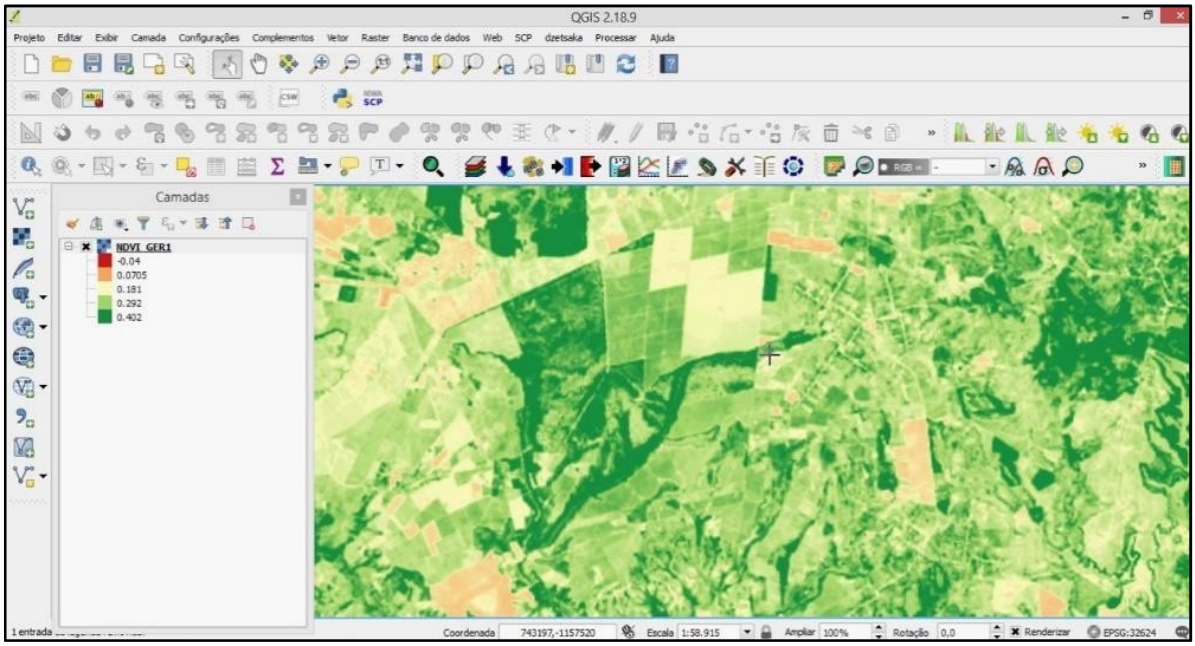

Рисунок 2 – Интерфейс QGIS

QGIS (Quantum GIS) представляет собой свободное кроссплатформенное программное обеспечение, предназначенное для работы с геопространственными данными. Данное приложение предоставляет пользователям широкий спектр возможностей по визуализации, редактированию и

анализу геоданных различных форматов, а также созданию картографических материалов и проведению пространственного анализа. С помощью QGIS можно выполнять следующие задачи:

1) импортировать, экспортировать и преобразовывать данные из различных форматов, включая Shapefile, GeoJSON, KML;

2) просматривать и редактировать геоданные на карте, включая точки, линии, полигоны, растровые изображения;

3) создавать, редактировать и стилизовать карты, добавляя элементы управления, легенды, компасы;

4) выполнять пространственный анализ, например, расчет расстояний, площадей, пересечений, объединений;

5) производить интерполяцию, классификацию, градацию и прочие операции со статистическими данными;

6) представлять данные в трехмерном виде и многое другое.

QGIS в настоящее время предлагает широкий спектр инструментов для анализа и обработки геопространственных данных, включая векторный и растровый анализ, выборку, геообработку, геометрию и управление базами данных. Кроме того, QGIS интегрирует в себя функциональность геоинформационной системы GRASS, предоставляя доступ к более чем 400 модулям, что значительно расширяет возможности приложения. Все функции анализа выполняются в фоновом режиме, позволяя пользователям продолжать работу без необходимости ожидания завершения обработки данных.

Одной из ключевых особенностей QGIS является его открытость и бесплатность, что делает приложение доступным для широкого круга пользователей, включая исследователей, студентов и специалистов в области геоинформационных технологий. Открытый исходный код QGIS позволяет сообществу разработчиков создавать и совершенствовать функциональность приложения, разрабатывая плагины, расширения и дополнительные инструменты. Благодаря активному сообществу пользователей и

разработчиков, QGIS постоянно развивается и адаптируется к новым задачам и требованиям в области анализа и обработки геопространственных данных.

УралГИСАгро [1]

УралГИСАгро представляет собой комплексное инфраструктурное решение, предназначенное для мониторинга, контроля, информационного сопровождения и принятия управленческих решений в сфере сельского хозяйства. Система использует пространственные отраслевые данные, полученные от муниципальных и региональных ведомств, для анализа текущей ситуации и прогнозирования развития агропромышленного комплекса.

Одной из ключевых функций УралГИСАгро является обеспечение межведомственного взаимодействия, что позволяет координировать работу различных организаций и ведомств, вовлеченных в процесс управления сельским хозяйством. Система предоставляет единую платформу для обмена информацией, что способствует повышению эффективности принятия решений и оперативности реагирования на возникающие проблемы. Функционал, который используется в УралГИСАгро, изображен на рисунке 3.

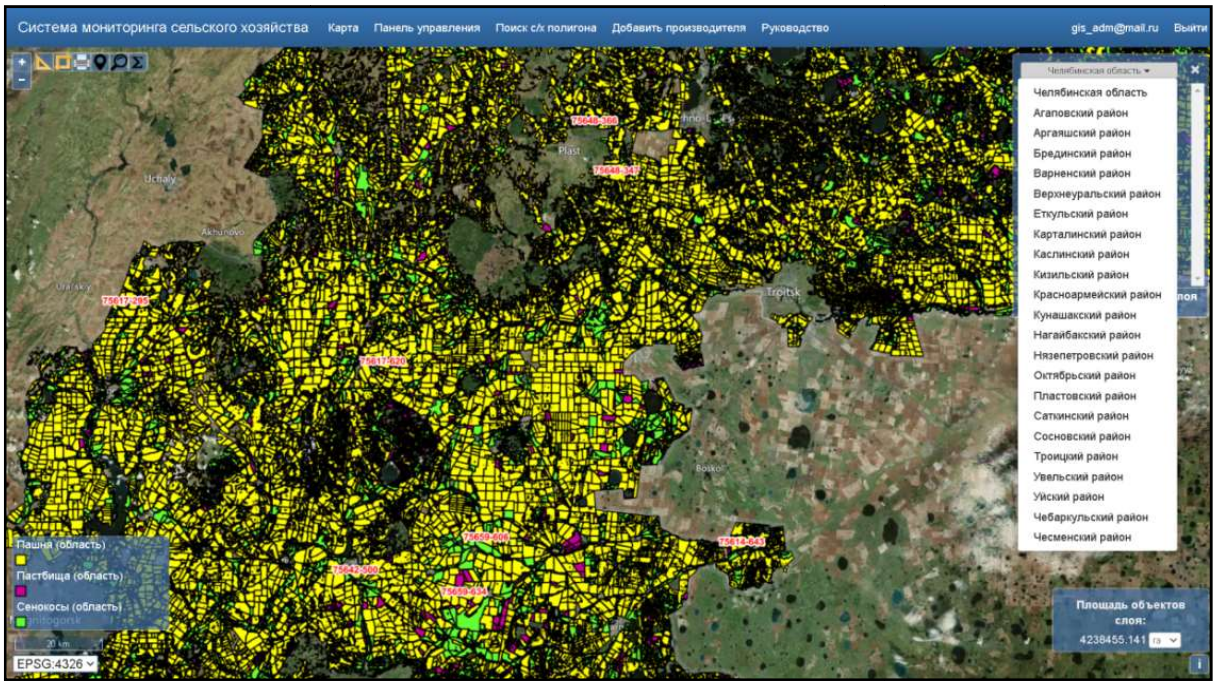

Рисунок 3 – Интерфейс УралГИСАгро

Сущность системы УралГИСАгро заключается в интеграции разнородной информации об объектах сельского хозяйства области в едином информационном пространстве. Это позволяет создать целостную картину агропромышленного комплекса региона, учитывая различные аспекты его функционирования. УралГИСАгро обеспечивает высокую степень визуализации данных, а также позволяет осуществлять эффективный мониторинг и контроль состояния отрасли. Таким образом, УралГИСАгро представляет собой мощный инструмент для информационной поддержки принятия решений в сфере управления сельским хозяйством на региональном уровне, обеспечивая комплексный подход к анализу и визуализации разнородных данных об объектах агропромышленного комплекса.

1.2. Обзор инструментов для реализации

Для реализации приложения необходимо использовать библиотеки, которые позволят создать графический интерфейс приложения, обеспечат возможности для чтения, обработки и визуализации данных в формате Tiff и shp, а также позволят создать функции, необходимые для осуществления анализа заболоченности.

Rasterio [13]

Библиотека для работы с растровыми данными в языке программирования Python. Она предоставляет инструменты для чтения, записи и обработки растровых изображений, таких как спутниковые изображения, карты высотности и другие гео- и космические данные. Одной из ключевых особенностей Rasterio является его способность обрабатывать различные форматы растровых данных, включая GeoTiff, JPEG, PNG и многие другие. Это обеспечивает универсальность при работе с разнообразными источниками данных.

Geopandas [4]

Geopandas – это библиотека для работы с геопространственными данными в языке программирования Python. Она предоставляет функцио-

нальность для удобной работы с геометрическими данными, такими как точки, линии и полигоны, интегрируя их с функциональностью библиотеки pandas для анализа данных. Библиотека Geopandas поддерживает различные форматы геопространственных данных, в том числе формат Shapefile, который используется для определения границ объекта, а также предоставляет возможность выполнения пространственных операций, таких как объединение, пересечение и разность геометрических объектов.

Tkinter [15]

Tkinter (Tcl/Tk интерфейс) является стандартной библиотекой для создания графических пользовательских интерфейсов в языке программирования Python. Она предоставляет простые инструменты для создания окон, виджетов и управления событиями, что делает ее подходящей для начинающих и для быстрого создания простых приложений. Tkinter известен своей простотой и легкостью использования.

PyTiff [11]

PyTiff – это библиотека на Python для работы с изображениями в формате TIFF. Эта библиотека предоставляет удобные инструменты для чтения, записи и обработки TIFF-изображений. PyTiff представляет собой полезную и эффективную библиотеку для работы с TIFF-изображениями на Python. Она идеально подходит для задач, связанных с обработкой и анализом графических данных, и может быть использована как для быстрых прототипов, так и для разработки сложных приложений в области обработки изображений.

NumPy [7]

NumPy – это основная библиотека на Python для выполнения научных вычислений и работы с многомерными массивами (включая матрицы), которая также является фундаментом для библиотек Pandas, Matplotlib, SciPy и TensorFlow. Она предоставляет мощные инструменты для работы с числовыми данными, алгебраическими операциями, статистикой, транс-

формациями данных и другими задачами, связанными с научными вычислениями.

GDAL [3]

GDAL представляет собой библиотеку с открытым исходным кодом, предназначенную для чтения и записи растровых и векторных геопространственных данных. Она предоставляет мощные инструменты для обработки и анализа различных типов геоданных, таких как спутниковые снимки, векторные слои и цифровые модели рельефа, и является общепризнанным стандартом в индустрии геоинформационных систем.

Matplotlib [6]

Matplotlib является одной из наиболее широко используемых библиотек для визуализации данных на языке Python. Она предоставляет гибкий и мощный инструментарий для построения различных типов графиков, диаграмм, линейных графиков, гистограмм, диаграмм рассеяния, 3D-графиков. Matplotlib позволяет тонко настраивать все аспекты визуализации, такие как цвета, стили линий, шрифты и размеры, что дает возможность отобразить информацию наиболее наглядно и удобно.

Folium [2]

Folium – это библиотека Python, которая позволяет создавать интерактивные карты с использованием библиотеки leaflet.js. Она предоставляет простой и удобный API для визуализации геопространственных данных в веб-браузере. С помощью Folium можно создавать карты с различными слоями, маркерами и всплывающими окнами. Библиотека поддерживает различные типы карт, включая OpenStreetMap, Mapbox и Stamen, а также позволяет настраивать стили и цвета элементов карты. Folium широко используется в области анализа данных и геопространственной визуализации, особенно в сочетании с библиотеками, такими как Pandas и GeoPandas. Она предоставляет удобный способ создания информативных и привлекательных и сохранения их в виде HTML-файлов для дальнейшего использования.

Выводы по первой главе

В данной главе была рассмотрена предметная область и проведен обзор существующих аналогов для анализа заболоченности водоемов. SentinelHub является одним из ведущих решений в этой области, предоставляя богатый функционал для работы с данными спутниковых снимков. Платформа обеспечивает доступ к таким индексам, как WVI, оптическим изображениям высокого разрешения и алгоритмам классификации, что позволяет проводить комплексный анализ состояния водных объектов.

Другим значимым аналогом является свободное и кроссплатформенное приложение QGIS, которое предоставляет широкие возможности для работы с геоданными и проведения пространственного анализа. QGIS поддерживает множество форматов данных, включая векторные и растровые, а также предлагает обширный набор инструментов для их обработки, визуализации и анализа.

Кроме того, был рассмотрен сайт УралГИСАгро – комплексное решение для мониторинга, контроля, обработки и хранения геоинформационных данных в сфере сельского хозяйства. Система обеспечивает интеграцию разнородной информации в едином пространстве, что позволяет эффективно решать задачи управления и принятия решений в агропромышленном комплексе.

В главе также были рассмотрены основные библиотеки, необходимые для создания приложения, работающего с геопространственными данными в Python. Обзор библиотек Geopandas, Rasterio, Pytiff, GDAL, Numpy, Folium, Matplotlib и Tkinter показал, как эти инструменты взаимодействуют и дополняют друг друга, обеспечивая полный стек функциональности для геоинформационного анализа и визуализации.

Обобщая, целью разработки нового приложения должно быть создание инструмента с функционалом, позволяющим проводить операции по анализу данных, а также имеющим интуитивно понятный и эффективный интерфейс для работы с геопространственными данными.

2. ПРОЕКТИРОВАНИЕ

2.1. Выявление функциональных и нефункциональных требований, построение диаграммы вариантов использования

Функциональные требования – это требования, которые определяют действия, которые должна выполнять система, без учета ограничений, связанных с ее реализацией, то есть определяют поведение системы в процессе обработки информации. Нефункциональные требования – это условия, при которых продукт должен работать, и качества, которыми он должен обладать (например, производительность, надежность, масштабируемость).

Функциональные требования

К разрабатываемой системе имеются следующие функциональные требования:

1) система должна обеспечивать сбор данных о заболоченности водоемов;
2) система должна предоставлять визуализацию исходных данных;
3) система должна обеспечить возможность проведения статистических данных.

Нефункциональные требования

К разрабатываемой системе имеются следующие нефункциональные требования:

1) система должна быть стабильной и надежной;
2) система должна работать на соверменных версиях ОС Windows;
3) программный код системы должен быть кратким и лаконичным.

Диаграмма вариантов использования

Для проектирования системы был использован язык графического описания для объектно-ориентированного программирования UML. Была построена модель взаимодействия внешнего актера с системой приложения в виде диаграммы вариантов использования. Диаграмма была построена с помощью PlantUML [8], инструментом с открытым исходным кодом,

который позволяет быстро создавать диаграммы на основе языка разметки. В ходе анализа требований к разрабатываемой системе были выявлены варианты использования, показанные на рисунке 4.

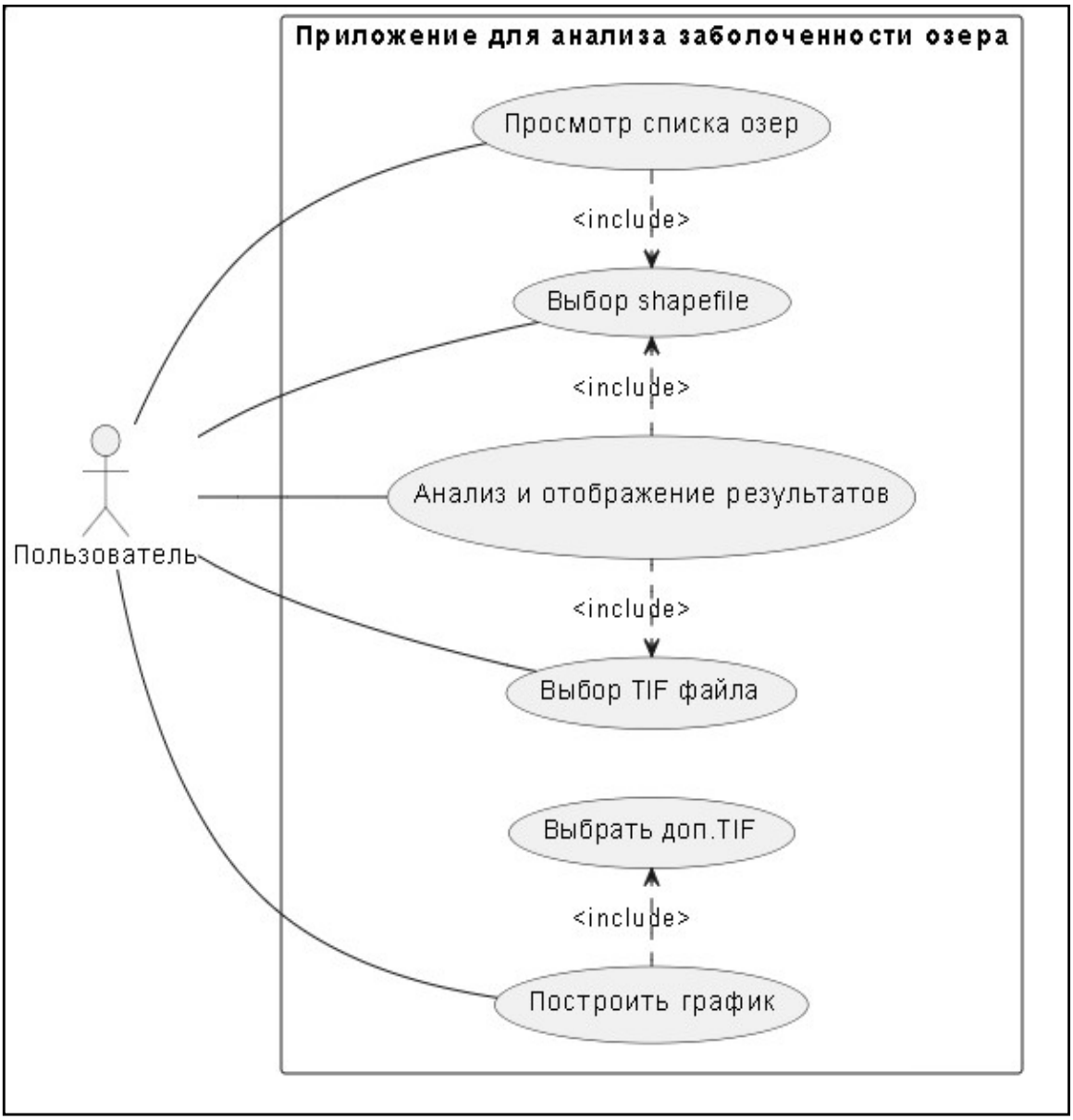

Рисунок 4 – Use-case диаграмма системы

Выводы по второй главе

В ходе проектирования были определены функциональные требования к системе. На основе этих требований была построена диаграмма вариантов использования, определены основные актеры, взаимодействующие с системой.

3. РЕАЛИЗАЦИЯ

Программные средства реализации

Для реализации программной части системы был выбран язык программирования Python. Разработка велась в среде программирования PyCharm. Для реализации компонентов были использованы следующие библиотеки: Geopandas, Rasterio, Numpy, GDAL, Tkinter, Matplotlib.

3.1. Разработка модуля Genmap

Модуль Genmap был разработан для фильтрации объектов, полученных из shape-файлов, так как некоторые файлы данного формата, полученные от УралГИСАгро, содержали некорректные данные, которые могли бы утяжелить процесс поиска необходимой для осуществления анализа информации. Данный модуль создает интерактивную карту, на которую загружается shape-файл, каждому объекту присваивается идентификатор, после чего карта сохраняется в html формате и может быть использована для выбора необходимых объектов. Реализация модуля Genmap приведена в листинге 1.

Листинг 1 – Реализация модуля Genmap

```
# Загрузка shape-файла
shapefile1 = '../maps/1/shapes/ag_vod.shp'
shapefile2 = '../maps/2/shapes/arg_vod.shp'
gdf = gpd.read_file(shapefile2)

print(gdf.columns)

gdf_centroid = gdf.centroid.to_crs(epsg=4326)
map_center = gdf_centroid.unary_union.centroid

# Создание интерактивной карты
interactive_map = folium.Map(location=[map_center.y, map_center.x],
zoom_start=6)

# Добавление shape-форм на карту
folium.GeoJson(gdf).add_to(interactive_map)

# Функция для обработки кликов на shape-формы
def on_click(feature, **kwargs):
    print(feature)
    shape_name = feature['properties']['name']
    print(f"Выбрана shape-форма: {shape_name}")
    # Дополнительные действия с выбранной shape-формой

# Добавление обработчика кликов на shape-формы
```

```
folium.GeoJson(
    gdf,
    name='Shapes',
    style_function=lambda feature: {
        'fillColor': 'blue',
        'color': 'black',
        'weight': 2,
        'fillOpacity': 0.7
    },
    highlight_function=lambda feature: {
        'fillColor': 'red',
        'color': 'red',
        'weight': 3,
        'fillOpacity': 0.7
    },
    tooltip=folium.GeoJsonTooltip(
        fields=['id'],
        aliases=['ID:'],
        localize=True,
        sticky=False
    ),
).add_to(interactive_map)

# Отображение интерактивной карты
interactive_map.save('interactive_map.html')
```

Результат работы модуля Genmap изображен на рисунке 5.

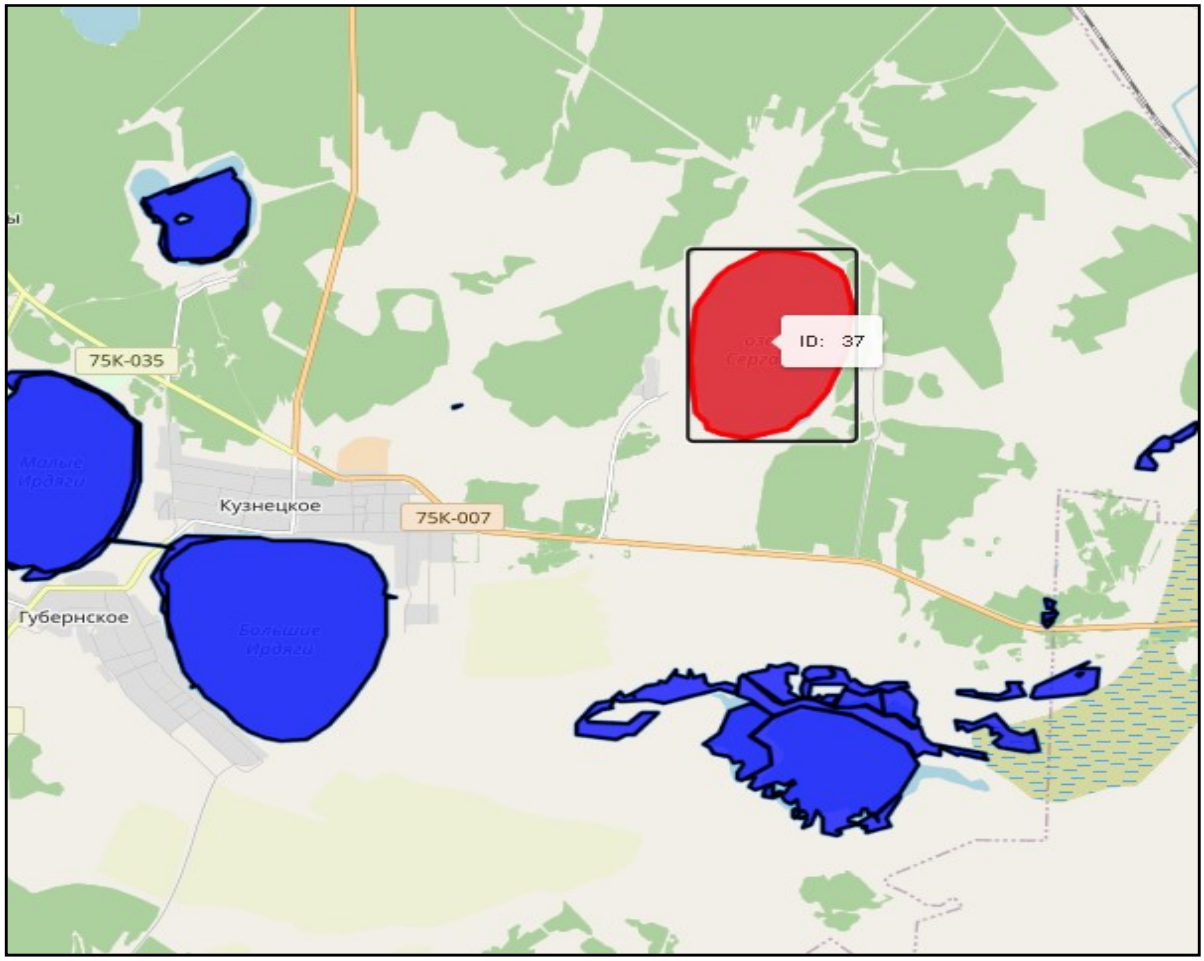

Рисунок 5 – Интерактивная карта Genmap

3.2. Разработка компонента GUI

Для реализации компонента GUI было разработано четыре модуля: `ContentFrame`, `ListFrame`, `TifLayout` и `GraphicsBuilder`. Модуль `ListFrame` определяет методы `change_shape_name`, `load_shapes` и `on_shape_id_change`, которые обновляют содержимое при изменении файла формы, конфигурации и смене shape-файла соответственно. Также в данном модуле определяется метод `read_shapes`, который считывает формы из файла конфигурации и возвращает список форм.

Реализация модуля ListFrame приведена в листинге 2.

Листинг 2 – Реализация модуля `ListFrame`

```
class ListFrame(tk.Frame):
    def __init__(self, parent, data: Data, shapes_config_path: str,
shapes_name: str = ""):
        super().__init__(parent)
        self.data = data
        self.shapes_config_path = shapes_config_path

        self.list_box = tk.Listbox(self, height=10)
        self.list_box.grid(row=0, column=0, sticky='nsew')

        self.scroll = ttk.Scrollbar(
            self, orient=tk.VERTICAL, command=self.list_box.yview)
        self.scroll.grid(row=0, column=1, sticky='nse')

        if len(shapes_name) > 0:
            self.load_shapes(shapes_name)

        self.data.shapefile_path.trace_add('write', self.change_shape_name)
        self.list_box.bind('<<ListboxSelect>>',
                           self.on_shape_id_change)  # on id change
        self.list_box.configure(yscrollcommand=self.scroll.set)

    def change_shape_name(self, *args) -> None:
        filename_without_ext = os.path.splitext(
            os.path.basename(self.data.shapefile_path.get()))[0]
        self.load_shapes(filename_without_ext)

    def load_shapes(self, shapes_name: str) -> None:
        self.list_box.delete(0, tk.END)
        self.shapes = self.read_shapes(shapes_name)
        for shape in self.shapes:
            self.list_box.insert(tk.END, shape)
        self.data.set_selected_id(self.shapes[0])

    def on_shape_id_change(self, event):
        selected_item = self.list_box.get(self.list_box.curselection())
        self.data.set_selected_id(selected_item)

    def read_shapes(self, shapes_name: str) -> list[int]:
        with open(self.shapes_config_path, "r") as f:
            data = json.load(f)
```

```
        shapes = data.get(shapes_name)

        if isinstance(shapes, list):
            return shapes
        else:
            print("Поле 'arg_vod' не является массивом")
            return []
```

Модуль `ContentFrame` определяет размеры и расположение изображений, которые будут отображаться в приложении, меток, которые привязаны к числовым значениям, полученных из расчетов. Также определяется метод `on_common_change`, который обновляет значения заболоченности.

Реализация модуля `ContentFrame` приведена в листинге 3.

Листинг 3 – Реализация модуля ContentFrame

```
class ContentFrame(tk.Frame):
    img_width = 150
    img_height = 150

    def __init__(self, parent, data: Data):
        super().__init__(parent)
        self.data = data

        self.tif_left = TifLayout(self, data, 0)
        self.tif_right = TifLayout(self, data, 1)

        self.common_frame = ttk.Frame(self, borderwidth=2, relief="solid")
        self.common_frame.grid(row=9, columnspan=2, pady=(10, 0),
sticky="ew")

self.common_ndvi = ttk.Label(
    self.common_frame, text="Разница процента заболочености: « anc-
hor="center")
    self.common_ndvi.grid(row=0, column=0, padx=10, pady=0, sticky="nsew")
    self.common_low = ttk.Label(self.common_frame, text="Разница Low: ",
anchor="center")
    self.common_low.grid(row=1, column=0, padx=10, pady=0, sticky="nsew")
    self.common_high = ttk.Label(self.common_frame, text="Разница High: ",
anchor="center")
    self.common_high.grid(row=3, column=0, padx=10, pady=0, sticky="nsew")

    self.common_frame.grid_rowconfigure(0, weight=1)
    self.common_frame.grid_rowconfigure(1, weight=1)
    self.common_frame.grid_columnconfigure(0, weight=1)

    self.data.common.trace_add('write', self.on_common_change)

def on_common_change(self, *args) -> None:
    values = ast.literal_eval(self.data.common.get())

    def format_number(x):
        return "{:.2f}%".format(x)
    self.common_ndvi.configure(
        text=f"Разница процента заболочености: {for-
mat_number(values['ndvi'])}")
    self.common_low.configure(
        text=f"Разница LOW: {format_number(values['low'])}")
```

```
    self.common_mid.configure(
    self.common_high.configure(
        text=f"Разница HIGH: {format_number(values['high'])}")
```

Модуль `TifLayout` определяет методы `on_ndvi_change`, `on_shape_id_change`, `on_image_change`, `change_tif` и `open_fullscreen`, которые обновляют содержимое и функциональность при изменении значений NDVI, смене идентификатора формы, изменении изображения, выборе нового TIF-файла и открытии изображения на полный экран соответственно.

Реализация модуля `TifLayout` приведена в листинге 4.

Листинг 4 – Реализация модуля `TifLayout`

```
def on_ndvi_change(self, *args) -> None:
    values = ast.literal_eval(self.data.ndvis[self.idx].get())

    self.text_ndvi.configure(text=f"{self.ndvi_name}: {for-
mat_number(values['ndvi'])}")
    self.text_low.configure(text=f"{self.low_name}: {for-
mat_number(values['low'])}")
    self.text_mid.configure(text=f"{self.mid_name}: {for-
mat_number(values['mid'])}")
    self.text_high.configure(text=f"{self.high_name}: {for-
mat_number(values['high'])}")

def on_shape_id_change(self, *args) -> None:
    self.data.change_image(self.idx)
    self.on_image_change()
def on_image_change(self, *args) -> None:
    self.photo = ImageTk.PhotoImage(self.data.get_image(self.idx))
    self.image.configure(image=self.photo)

def change_tif(self) -> None:
    file_types = [('TIF Files', '*.tif')]
    file_path = filedialog.askopenfilename(filetypes=file_types)
    if file_path:
        self.data.set_tif(file_path, self.idx)

def open_fullscreen(self) -> None:
    img = self.data.get_image(self.idx)
    if img is None:
        return

    self.fullscreen = tk.Toplevel()
    self.fullscreen.geometry(f"{img.width}x{img.height}")

    label = ttk.Label(self.fullscreen)
    label.pack(fill='both', expand=True)

def resize_image(event):
    new_width = event.width
    new_height = event.height
    resized_img = img.resize((new_width, new_height))
    self.fullscreen_photo = ImageTk.PhotoImage(resized_img)
```

```
label.configure(image=self.fullscreen_photo)

self.fullscreen.bind("<Configure>", resize_image)
self.fullscreen.mainloop()
```

Модуль `GraphicsBuilder` был реализован в отдельном окне так, чтобы не нарушать логику и структуру основного окна. Данный модуль определяет следующие методы: `choose_shapefile`, `choose_tif`, `on_id_change`, `on_graphics_build` и `read_shapes`, которые обеспечивают функциональность для выбора shape-файлов, выбора TIF-файлов, изменения выбранного идентификатора формы, построения графика и чтения идентификаторов форм из файла конфигурации соответственно.

Реализация модуля `GraphicsBuilder` приведена в листинге 5.

Листинг 5 – Реализация модуля `GraphicsBuilder`

```
def on_graphics_build(self, *args):
    shape_path = self.fileshape_path
    shape_id = int(self.selected_id)

    ndvis = []
    for tif_path in self.tifs:
        img = load_image(tif_path, shape_path, shape_id)

        treshold = 0.26
        ndvi_mask = img > treshold
        valid_img = ~np.isnan(img)
        count = np.sum(ndvi_mask[valid_img])
        total_count = np.sum(valid_img)
        ndvi = count / total_count
        ndvis.append(ndvi*100)


    x = np.arange(1, len(ndvis) + 1)
    y = np.array(ndvis)

    _, ax = plt.subplots()
    ax.plot(x, y, color='blue', marker='o', label="NDVI")
    ax.set_xticks(x)
    ax.set_yticks(y)
    ax.set_xlabel('Номер точки')
    ax.set_ylabel('NDVI')
    ax.tick_params('x', colors='blue')
    ax.legend()

    plt.show()
```

Таким образом, после успешной реализации компонента GUI, главное окно приложения примет вид, изображенный на рисунке 6.

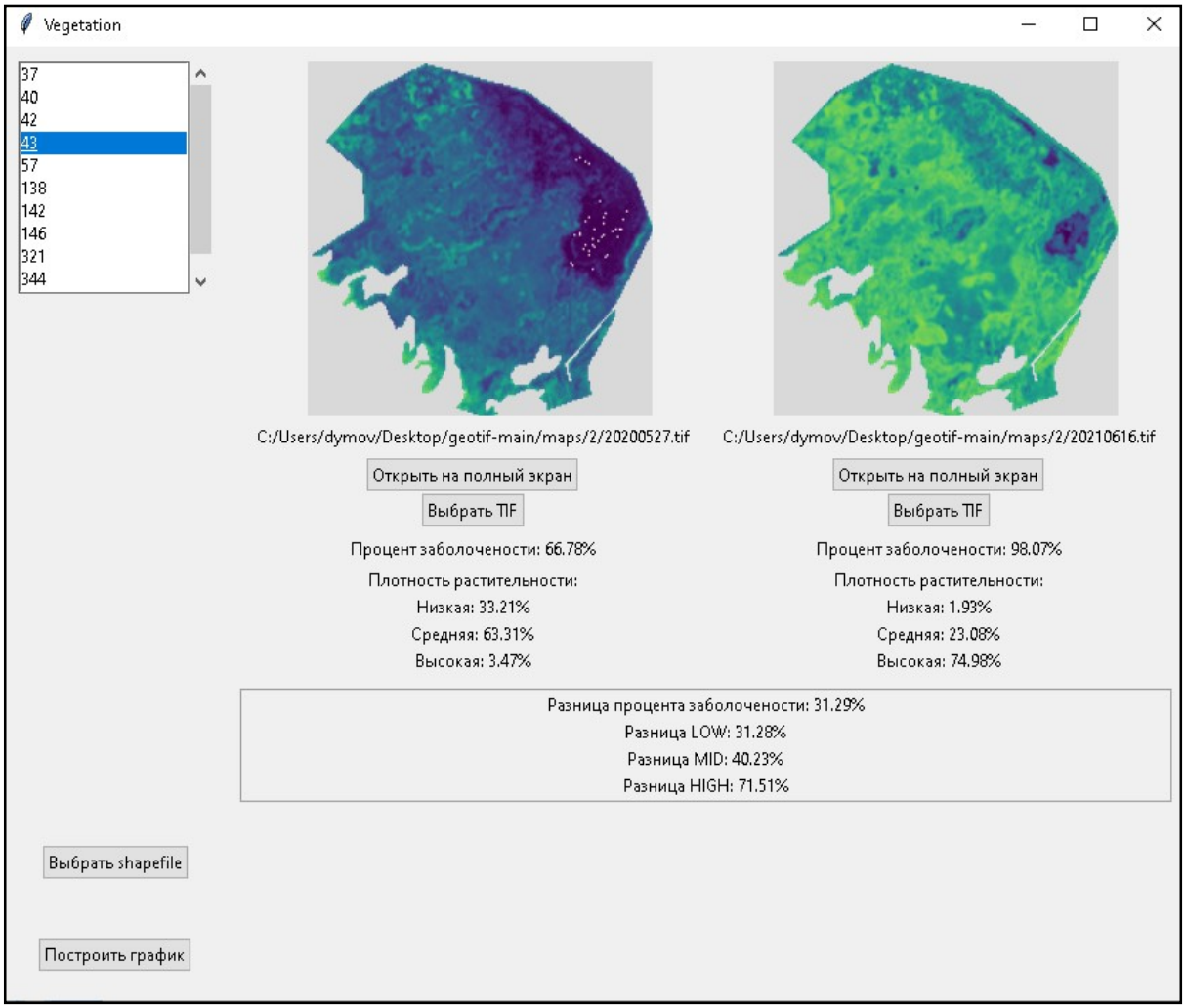

Рисунок 6 – Главное окно приложения

Окно выбора и настройки данных для построения графика и макет графика изображены на рисунке 7.

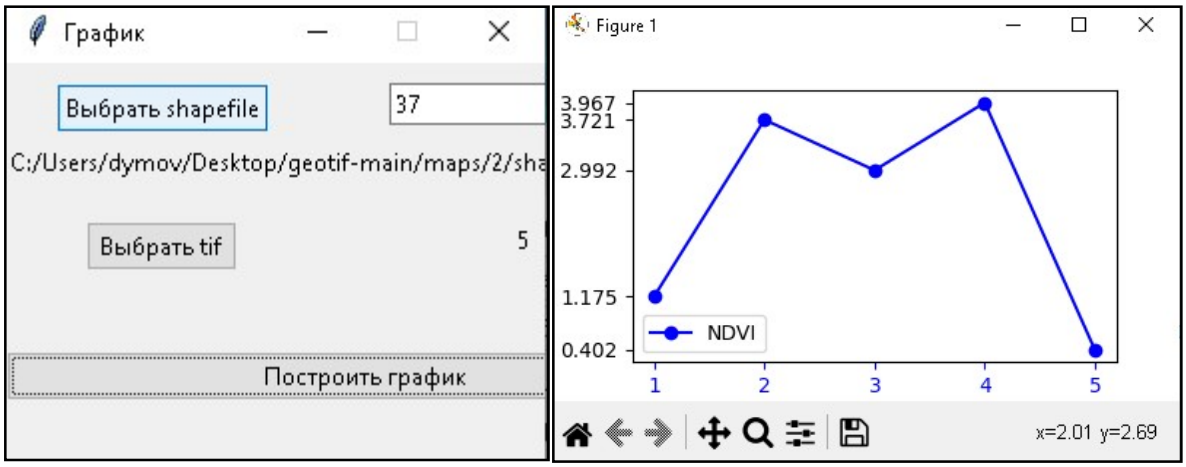

Рисунок 7 – Окно выбора и настройки данных для построения графика и макет графика

3.3. Реализация модуля Data

Реализация модуля `Data` обеспечивает централизованное управление данными приложения, включая загрузку изображений, вычисление значений NDVI и хранение различных переменных, необходимых для работы приложения. Этот модуль позволяет легко получать доступ к данным из других частей приложения и обновлять их по мере необходимости. Данный модуль определяет класс `Data`, который инкапсулирует различные переменные и методы для работы с данными с помощью метода `__init__`. Метод инициализации приведен в листинге 6.

Листинг 6 – Реализация метода `__init__`

```
def __init__(self):
    self.selected_id = tk.IntVar(value=0, name="Selected Item")
    self.images = [None]*2
    self.photos = [tk.Variable(name="Photo 1"),tk.Variable(name="Photo 2")]
    self.shapefile_path = tk.StringVar(value="", name="SHAPEFILE PATH")
    self.shape_id = tk.IntVar(value=0, name="SHAPE ID")
    self.tif_images_pathes = [tk.StringVar(
      value="", name="TIF 1"), tk.StringVar(value="", name="TIF 2")]
    self.ndvis = [tk.Variable(value={
      "ndvi": 0,
      "low": 0,
      "mid": 0,
      "high": 0
    }, name="NDVI 1"), tk.Variable(value={
      "ndvi": 0,
      "low": 0,
      "mid": 0,
      "high": 0
    }, name="NDVI 2")]

    self.common = tk.Variable(value={
      "ndvi": 0,
      "low": 0,
      "mid": 0,
      "high": 0
    })
```

Также в данном классе находятся расчетные формулы, по которым осуществляется обновление индекса NDVI и плотности растительности, а также их сохранение. Код данной части модуля приведен в листинге 7.

Листинг 7 – Код расчетной части приложения

```
 img = load_image(tif_path, shape_path, shape_id)

    treshold = 0.26
    mid_treshold = 0.55
    ndvi_mask = img > treshold
    low_mask = img < treshold
```

```
mid_mask = (img > treshold) & (img < mid_treshold)
high_mask = img > mid_treshold
valid_img = ~np.isnan(img)
count = np.sum(ndvi_mask[valid_img])
total_count = np.sum(valid_img)
ndvi = count / total_count
low = np.sum(low_mask[valid_img]) / total_count
mid = np.sum(mid_mask[valid_img]) / total_count
high = np.sum(high_mask[valid_img]) / total_count

pil_image = rasterio_img_to_pil(img)

self.images[idx] = pil_image
self.photos[idx].set(ImageTk.PhotoImage(pil_image))

self.ndvis[idx].set({
  "ndvi": ndvi,
  "low": low,
  "mid": mid,
  "high": high,
})

values_dict = [ast.literal_eval(v.get()) for v in self.ndvis]
ndvis = [v["ndvi"] for v in values_dict]
lows = [v["low"] for v in values_dict]
mids = [v["mid"] for v in values_dict]
highs = [v["high"] for v in values_dict]
self.common.set({"ndvi": max(ndvis) - min(ndvis),
  "low": max(lows) - min(lows),
  "mid": max(mids) - min(mids),
  "high": max(highs) - min(highs),
})
```

3.4. Реализация модуля img

Модуль `img` предоставляет функциональность для загрузки, обработки и преобразования геопространственных данных, таких как растровые изображения и шейп-файлы.

Внутри модуля `img` определяются две основные функции: `load_image` и `rasterio_img_to_pil`.

Функция `load_image` принимает путь к файлу формата GeoTIFF (`tif_path`), путь к shape-файлу (`shape_path`) и идентификатор геометрии (`shape_id`) в качестве аргументов. Она загружает геопространственные данные с использованием библиотек GeoPandas и Rasterio, извлекает конкретную геометрию из shape-файла на основе заданного идентификатора и выполняет операцию маскирования растрового изображения с использованием этой геометрии. Результатом является маскированное растровое изо-

бражение, соответствующее заданной геометрии. Код модуля `img` приведен в листинге 8.

Листинг 8 – Код модуля `img`

```
def load_image(tif_path: str, shape_path: str, shape_id: int) -> np.array:
    shapefile = gpd.read_file(shape_path)
    with rasterio.open(tif_path) as src:
        raster_crs = src.crs
        shapefile = shapefile.to_crs(raster_crs)
        shape = shapefile.loc[shapefile['id'] == shape_id]

        out_image, out_transform = rasterio.mask.mask(
            src, shape.geometry, crop=True, filled=True, nodata=np.nan)
        out_meta = src.meta.copy()
        out_meta.update({"driver": "GTiff",
                         "height": out_image.shape[1],
                          "width": out_image.shape[2],
                          "transform": out_transform})

        out_image[np.isnan(out_image)] = np.nan

        return out_image[0]


def rasterio_img_to_pil(img: np.array, size: tuple[int, int] = (250, 250))
-> ImageTk.Image.Image:
    cmap = plt.get_cmap('viridis')
    colored_data = cmap(img)

    pil_image = Image.fromarray(
        (colored_data * 255).astype('uint8'), mode='RGBA').resize(size)

    return pil_image


if __name__ == "__main__":
    tif_path = "maps/2/20200623.tif"
    shape_path = "maps/2/shapes/arg_vod.shp"
    shape_id = 43

    root = tk.Tk()
    root.geometry("800x600")

    img = load_image(tif_path, shape_path, shape_id)
    pil_image = rasterio_img_to_pil(img)
    photo = pil_image

    image_label = ttk.Label(root)
    image_label.configure(image=photo)
    image_label.pack()

    root.mainloop()
```

Выводы по третьей главе

Для реализации программной части системы был выбран язык программирования Python, а разработка велась в среде программирования

PyCharm. Использовались библиотеки Geopandas, Rasterio, Numpy, GDAL, Tkinter и Matplotlib для реализации различных компонентов системы.

Модуль `Genmap` был разработан для фильтрации объектов, полученных из shape-файлов, и создания интерактивной карты, на которую загружается shape-файл. Каждому объекту присваивается идентификатор, и карта сохраняется в формате HTML для дальнейшего выбора необходимых объектов.

Компонент GUI был реализован с помощью четырех модулей: `ContentFrame`, `ListFrame`, `TifLayout` и `GraphicsBuilder`. Эти модули определяют методы для обновления содержимого и функциональности при изменении различных параметров, таких как значения NDVI, идентификаторы форм, изображения и выбор файлов.

Модуль `Data` обеспечивает централизованное управление данными приложения, включая загрузку изображений, вычисление значений NDVI и хранение переменных, необходимых для работы приложения. Он инкапсулирует различные переменные и методы для работы с данными.

Модуль `img` предоставляет функциональность для загрузки, обработки и преобразования геопространственных данных, таких как растровые изображения и shape-файлы. Он содержит функции `load_image` и `rasterio_img_to_pil` для загрузки и преобразования данных.

В целом, глава «Реализация» описывает процесс разработки различных модулей и компонентов системы с использованием языка программирования Python и соответствующих библиотек. Каждый модуль выполняет определенные функции, такие как фильтрация объектов, создание интерактивной карты, управление данными и обработка геопространственных данных. Эти модули взаимодействуют друг с другом для обеспечения функциональности системы в целом.

4. ТЕСТИРОВАНИЕ

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения в определенных задачах решать задачи, необходимые пользователям.

Используя методологию функционального тестирования, была проверена работа мобильного приложения. В таблице 1 представлены результаты тестирования на функциональность.

Таблица 1 – Функциональное тестирование

Название теста	**Шаги**	**Ожидаемый результат**	**Тест пройден?**
Запуск приложения	1. Установить необходимые библиотеки 2. Запустить файл App.py 3. Получить вывод на экран главного окна приложения	Успешный запуск приложения и вывод главного окна на экран	Да
Построение интерактивной карты	1. Запустить файл genmap.py 2. Получить результаты работы программы 2. Открыть созданный файл interactive_map.html	Получение интерактивной карты с водоемами	Да
Загрузка файлов .tiff, .shp и проверка на работоспособность	1.Загрузить файлы .tiff, .shp в корневую папку приложения 2. Выбрать .tiff изображения внутри приложения 3. Выбрать .shp файлы внутри приложения 4.Выбрать желаемый водоем 5. Получить результаты работы программы	Успешная загрузка файлов и получение результатов работы программы	Да
Построение графика	1. Выбрать пункт «Построить график» 2. Выбрать .tiff изображения внутри приложения 3. Выбрать .shp файлы внутри приложения 3.Выбрать желаемый водоем 4.Вывести график на экран	Успешный вывод на экран графика с индексом NDVI	Да

Окончание таблицы 1

Название теста	Шаги	Ожидаемый результат	Тест пройден?
Построение графика, используя изображения разных регионов	1. Выбрать пункт «Построить график» 2. Ввести некорректные .tiff, .shp файлы 3.Выбрать желаемый водоем 4.Вывести график на экран	Вывод ошибки «Input shapes do not overlap raster.»	Да
Выбор разных .tiff файлов	1. Выбрать .tiff изображения из директории ag и arg 2. Выбрать .tiff изображения из директории arg 3. Выбрать .shp файлы внутри приложения 4.Выбрать желаемый водоем 5. Получить результаты работы программы	Вывод ошибки «Input shapes do not overlap raster.»	Да
Выбор некорректного .shp файла	1. Выбрать .shp файл из папки ag 2. Выбрать .tif файлы из папки arg 3.Выбрать желаемый водоем 4. Получить результаты работы программы	Вывод ошибки «Input shapes do not overlap raster.»	Да

Выводы по четвертой главе

В четвертой главе было проведено функциональное тестирование разработанного приложения. Результаты функционального тестирования показали, что приложение успешно выполняет все основные задачи, для которых оно было разработано, и способно корректно обрабатывать различные входные данные, выдавая соответствующие сообщения об ошибках.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано настольное приложение на языке Python для анализа заболоченности водоемов. При этом были решены задачи:

1) проведен обзор аналогичных систем;
2) рассмотрены библиотеки, необходимые для реализации приложения;
3) разработана структура и функциональная часть приложения;
4) разработано Python приложение, обеспечивающее анализ данных о заболоченности водных объектов;
5) реализован графический интерфейс для пользовательского взаимодействия;
6) проведено тестирование приложения;
7) приложение опубликовано на GitHub.

В ходе работы были освоены методы разработки приложений на языке Python, методы создания графического интерфейса и методы чтения, анализа и визуализации данных с файлов формата Tiff.

В дальнейшем планируется расширить функционал приложения, улучшить уже существующие методы расчета заболоченности водоемов, а также доработать графический интерфейс, сделав его более удобным в использовании.

ЛИТЕРАТУРА

1. УралГИСАгро. [Электронный ресурс] URL: https://uralgis.ru/ru/node/233/ (дата обращения: 09.02.2024 г.).

2. Folium. [Электронный ресурс] URL: https://python-visualization.github.io/folium/ (дата обращения: 10.02.2024 г.).

3. GDAL. [Электронный ресурс] URL: https://gdal.org/tutorials/ (дата обращения: 07.02.2024 г.).

4. GeoPandas. [Электронный ресурс] URL: https://geopandas.org/en/stable/ (дата обращения: 10.02.2024 г.).

5. Git. [Электронный ресурс] URL: https://git-scm.com/ (дата обращения: 10.02.2024 г.).

6. Matplotlib. [Электронный ресурс] URL: https://matplotlib.org/ (дата обращения: 10.02.2024 г.).

7. NumPy. [Электронный ресурс] URL: https://numpy.org/ (дата обращения: 10.02.2024 г.).

8. PlantUML. [Электронный ресурс] URL: https://plantuml.com/ru/ (дата обращения: 08.02.2024 г.).

9. Python. [Электронный ресурс] URL: https://www.python.org/ (дата обращения: 10.02.2024 г.).

10. PythonGIS. [Электронный ресурс] URL: https://pythongis.org/ (дата обращения: 08.02.2024 г.).

11. PyTiff. [Электронный ресурс] URL: https://pytiff.readthedocs.io/en/master/ (дата обращения: 10.02.2024 г.).

12. QGIS. [Электронный ресурс] URL: https://qgis.org/ru/site/ (дата обращения: 10.02.2024 г.).

13. Rasterio. [Электронный ресурс] URL: https://rasterio.readthedocs.io/en/stable/ (дата обращения: 10.02.2024 г.).

14. SentinelHub. [Электронный ресурс] URL: https://www.sentinel-hub.com/ (дата обращения: 08.02.2024 г.).

15. Tkinter. [Электронный ресурс] URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 10.02.2024 г.).