**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет «Информационных систем и технологий»

Кафедра «Интеллектуальных систем автоматизации и управления»

|  |  |
| --- | --- |
| Направление подготовки: | 09.03.02 – Информационные системы и технологии |
| Направленность (профиль): | Системное и прикладное программирование информационных систем |

**Отчет**

|  |
| --- |
| **о прохождении технологической (проектно-технологической) практики** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Выполнил студент группы | | ИСТ-261 |
|  |  |  | | |
|  |  | *Фамилия И. О.* | | |
|  |  | Руководитель практики от профильной организации | д.т.н., профессор | |
| *оценка* |  |  | *уч. степень, уч. звание* | |
|  |  | зав. каф. ИСАУ, Верхова Г.В. | | |
| *дата, подпись* |  | *должность, Фамилия И. О.* | | |
|  |  | Руководитель практики от кафедры | д.т.н., профессор | |
| *оценка* |  |  | *уч. степень, уч. звание* | |
|  |  | зав. каф. ИСАУ, Верхова Г.В. | | |
| *дата, подпись* |  | *должность, Фамилия И. О.* | | |

Оглавление

[**Техническое задание 3**](#_Toc172610352)

[**Введение 4**](#_Toc172610353)

[**Отчёт о выполненной работе 6**](#_Toc172610354)

[**Введение 6**](#_Toc172610355)

[**Технологический стек 6**](#_Toc172610356)

[**Структура проекта 7**](#_Toc172610357)

[**Основные объекты системы 7**](#_Toc172610358)

[**Реляционная модель базы данных 9**](#_Toc172610359)

[**Преимущества реляционной модели данных 10**](#_Toc172610360)

[**Функциональные возможности 11**](#_Toc172610361)

[**Интерфейс API 12**](#_Toc172610362)

[**GeoObject: 12**](#_Toc172610363)

[**GeoObjectsClassifiers 14**](#_Toc172610364)

[**ParentChild 15**](#_Toc172610365)

[**Topology 16**](#_Toc172610366)

[**Aspect 16**](#_Toc172610367)

[**Classifier 17**](#_Toc172610368)

[**Работа с клиентской частью 18**](#_Toc172610369)

[**Пример использования API в клиентском приложении 18**](#_Toc172610370)

[**Документация и поддержка 19**](#_Toc172610371)

[**Заключение 20**](#_Toc172610372)

[**Список использованной литературы 21**](#_Toc172610373)

# Техническое задание

Ядро системы многоаспектного геоинформационного моделирования должно обеспечивать следующие функции:

* создание географически распределенных объектов;
* создание топологических отношений между географически распределенными объектами (геосистемами) на базе объектно-ориентированной модели топологических отношений как на логическом, так и на геометрическом уровнях;
* создание семантических отношений между геосистемами;
* поиск геосистем с учетом топологических и семантических отношений;
* расширяемую поддержку классификаторов геосистем;
* визуализацию геосистем и топологических отношений на ГИС;
* задание основных параметром геосистем и связей между ними (включая автоматическое изменение статуса у ассоциированных геосистемы (паспорт геосистемы);
* управление статусом геосистем и связей между ними (включая автоматическое изменение статуса у ассоциированных геосистем).

Тестирование и документация REST API выполняется с помощью Swagger. Кроссплатформенная реализация на основе последней версии платформы Net. Графический интерфейс должен быть интуитивно понятным, эстетически привлекательным и динамичным.

# Введение

Геоинформационные системы (ГИС) представляют собой комплекс программных и аппаратных средств, предназначенных для сбора, хранения, анализа, управления и визуализации пространственных (географических) данных. ГИС являются мощным инструментом, используемым в различных областях, таких как урбанистика, экология, сельское хозяйство, логистика и многие другие.

Целью данного проекта являлась разработка ядра геоинформационной системы (ГИС) с использованием языка программирования C# и технологии Entity Framework. ГИС является важным инструментом для анализа, управления и визуализации пространственных данных, что особенно актуально в областях, таких как урбанистика, экология, логистика и другие.

Ядро системы было спроектировано с учетом современных принципов разработки программного обеспечения, включая модульность, масштабируемость и удобство сопровождения. Основными компонентами ядра являются:

1. Слой данных (Data Layer): Использование Entity Framework для работы с базой данных. Это обеспечивает гибкость и эффективность при манипулировании данными, а также упрощает выполнение CRUD-операций (создание, чтение, обновление и удаление).
2. Бизнес-логика (Business Logic Layer): Реализация основных алгоритмов и логики работы системы. Этот слой отвечает за обработку данных и выполнение различных операций, таких как пространственный анализ и расчет маршрутов.
3. API-интерфейс (API Layer): Предоставление интерфейсов для взаимодействия с клиентскими приложениями. Реализация RESTful API для интеграции с внешними системами и приложениями.

Используемые технологии

1. C#: Основной язык программирования, используемый для разработки ядра системы. Обеспечивает высокую производительность и поддержку объектно-ориентированного программирования.
2. Entity Framework: ORM (Object-Relational Mapping) фреймворк, который значительно упрощает работу с базой данных и позволяет использовать LINQ для выполнения запросов.
3. SQL Server: В качестве базы данных выбран SQL Server, что обеспечивает надежное хранение и быстрый доступ к пространственным данным.

Основные функции

* Управление пространственными данными: Добавление, обновление и удаление географических объектов, таких как точки, линии и полигоны.
* Пространственный анализ: Выполнение операций пространственного анализа, включая буферизацию, пересечение и объединение объектов.

# Отчёт о выполненной работе

## Введение

В ходе разработки были реализованы следующие ключевые функции:

* Создание и управление географически распределенными объектами
* Установление топологических и семантических связей между объектами
* Поиск и визуализация геосистем
* Управление статусами объектов и их связей

Для взаимодействия с системой используется REST API, тестирование и документация которого осуществляются с помощью Swagger. Язык программирования — C#, платформа .NET.

Для разработки серверной части системы были использованы следующие технологии и инструменты:

## Технологический стек

**Языки программирования и платформы**

* **C#**: основной язык программирования, используемый для разработки логики приложения.
* **.NET**: платформа для разработки и выполнения приложений.

**Библиотеки и фреймворки**

* **Entity Framework**: ORM (object-relational mapping) для работы с базой данных.
* **Swagger**: инструмент для документирования и тестирования REST API.

**Инструменты**

* **Visual Studio**: интегрированная среда разработки (IDE) для C# и .NET.
* **PostgresSQL**: система управления базами данных, используемая для хранения данных.
* **Visual Studio Code**

## Структура проекта

Проект состоит из нескольких модулей, каждый из которых выполняет свою роль в общей архитектуре системы. Ниже приведено описание основных модулей и их структуры:

**GISServer.API**

* **Controller**: реализуют конечные точки API для взаимодействия с клиентами.
* **Service**: содержат бизнес-логику приложения.
* **Model**: представляют данные, передаваемые между клиентом и сервером.
* **Mapper**: отвечает за преобразование данных из одной формы в другую.

**GISServer.Domain**

* **Model**: представляют основные объекты предметной области.
* **Repository**: обеспечивают доступ к данным, хранящимся в базе данных.

**GISServer.Infrastructure**

* **Data**: реализация репозиториев для работы с базой данных.
* **Migrations**: управление схемой базы данных.

## Основные объекты системы

Основным объектом в нашей системе является **GeoObject**. Он представляет собой географически распределенный объект с различными свойствами и связями. Подробное описание класса GeoObject приведено ниже:

**Свойства класса GeoObject**

* **Id**: уникальный идентификатор объекта.
* **Name**: название объекта.
* **GeoNameId**: идентификатор географического названия (опционально).
* **Status**: статус объекта.
* **UpdateTime**: время последнего обновления объекта.
* **CreationTime**: время создания объекта.
* **GeoNameFeatureId**: идентификатор функции географического названия (опционально).
* **GeoNameFeature**: функция географического названия.
* **Geometry**: текущая версия геометрии объекта.
* **GeometryVersion**: список версий геометрии объекта.
* **GeoObjectInfo**: дополнительная информация об объекте.
* **ParentGeoObjects**: список родительских объектов.
* **ChildGeoObjects**: список дочерних объектов.
* **OutputTopologyLinks**: список исходящих топологических связей.
* **InputTopologyLinks**: список входящих топологических связей.
* **Aspects**: список аспектов объекта.

# Реляционная модель базы данных

Реляционная модель данных имеет ключевое значение в разработке системы многоаспектного геоинформационного моделирования (ГИС), поскольку она обеспечивает структурированный и организованный способ хранения, управления и доступа к данным. Модель реляционной базы данных представлен на рисунке 1.

  
*Рисунок 1 - Модель базы данных, составленной при помощи возможностей PostgreSQL.*

В вашем проекте использование реляционной модели данных через ORM (Object-Relational Mapping) Entity Framework дает несколько важных преимуществ:

## Преимущества реляционной модели данных

1. **Структурированное хранение данных**
   * Реляционная модель организует данные в таблицы, где каждая таблица состоит из строк и столбцов. Это позволяет четко определить и хранить различные сущности и их отношения.
2. **Целостность данных**
   * Реляционные базы данных обеспечивают целостность данных через механизмы первичных и внешних ключей, ограничений и транзакций. Это помогает поддерживать корректность и согласованность данных в системе.
3. **Мощные возможности запросов**
   * Реляционные базы данных используют язык SQL (Structured Query Language) для выполнения сложных запросов к данным. Это позволяет эффективно искать, фильтровать и агрегировать данные.
4. **Гибкость и масштабируемость**
   * Реляционная модель данных позволяет легко добавлять новые таблицы и отношения между ними, что делает систему более гибкой и масштабируемой.
5. **Поддержка сложных связей**
   * В вашем проекте реляционная модель позволяет устанавливать и управлять сложными топологическими и семантическими связями между геообъектами, обеспечивая необходимую функциональность для моделирования геосистем.

## Функциональные возможности

Серверная часть системы обеспечивает следующие функциональные возможности:

* 1. **Создание географически распределенных объектов**

Клиенты могут создавать новые объекты, отправляя POST-запросы на соответствующие конечные точки API. Пример запроса:

POST /api/GeoObject

{

"name": "Object\_1",

"geometry": { "borderGeocodes": "{\"type\":\"Polygon\",\"coordinates\":[[59.95775871067945,30.400375014205686],[59.95940320281114,30.40732626404227],[59.95565186733682,30.40837753330767]]}"},

"geoObjectInfo": {

"commonInfo": "string"

}

}

* 1. **Создание топологических и семантических отношений**

Клиенты могут устанавливать связи между объектами, используя соответствующие конечные точки API. Пример запроса для установления топологической связи:

POST /api/Topology

{

"geographicalObjectInId": "96bbdbcb-3ca4-4154-8ec5-fbbb9c0e7dbe",

"geographicalObjectOutId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

* 1. **Расширяемая поддержка классификаторов**

Система поддерживает добавление и использование различных классификаторов геосистем. Пример запроса для добавления классификатора:

POST /api/GeoObject/AddClassifier

{

"geoObjectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"classifierId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

* 1. **Управление параметрами и статусами геосистем**

Система позволяет задавать и изменять параметры объектов и их связей. Пример запроса для изменения параметров объекта:

PUT /api/GeoObject/{id}

{

"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "Измененный объект",

"status": "Изменённый статус",

"geometry":

{

"type": "Polygon", "coordinates": [[30.0, 50.0], [31.0, 51.0], [32.0, 52.0]]

}

}

## Интерфейс API

Для взаимодействия с системой используется REST API. Ниже приведены все конечные точки, их описание примеры:

### GeoObject:

GET /api/GeoObject – получение всех гео-объектов:

Пример результата запроса:

**{**

"id": "8e9af5bf-bb0a-46f3-af97-ffb17fe68b0f",

"name": "OBJECT\_1",

"geoNameId": null,

"status": 50,

"geometry": {

"id": "8e9af5bf-bb0a-46f3-af97-ffb17fe68b0f",

"borderGeocodes": "{\"type\":\"Polygon\",\"coordinates\":[[59.95775871067945,30.400375014205686],[59.95940320281114,30.40732626404227],[59.95565186733682,30.40837753330767]]}"

},

"geoObjectInfo": {

"id": "8e9af5bf-bb0a-46f3-af97-ffb17fe68b0f",

"commonInfo": "string",

"classifiers": []

},

"parentGeoObjects": [],

"childGeoObjects": [],

"outputTopologyLinks": [],

"inputTopologyLinks": [],

"aspects": []

}

GET /api/GeoObject/{id} – получение гео-объекта по его Id.

POST /api/GeoObject – создание гео-объекта:

Пример запроса:

{

"name": "Object\_1",

"geometry": { "borderGeocodes": "{\"type\":\"Polygon\",\"coordinates\":[[59.95775871067945,30.400375014205686],[59.95940320281114,30.40732626404227],[59.95565186733682,30.40837753330767]]}"},

"geoObjectInfo": {

"commonInfo": "string"

}

}

PUT /api/GeoObject/{id} – изменение гео-объекта:

Пример запроса:

{

"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "Измененный объект",

"status": "Изменённый статус",

"geometry":

{

"type": "Polygon", "coordinates": [[30.0, 50.0], [31.0, 51.0], [32.0, 52.0]]

}

}

DELETE /api/GeoObject/{id} – удаление гео-объекта.

POST /api/GeoObject/AddClassifier – добавление классификатора к объекту:

Пример запроса:

{

"geoObjectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"classifierId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"}

DELETE /api/GeoObject/DeleteClassifier – удаление связи между гео-объектом и классификатором.

GET /api/GeoObject/GetAspects/{id} – получение аспектов гео-объекта:

Пример результата запроса:

{

"id": "8ccbe92b-ed0f-499c-847a-61b384ce88bb",

"type": "Aspect\_1",

"code": null,

"endPoint": null,

"commonInfo": null,

"geographicalObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa"

}

POST /api/GeoObject/AddAspect – добавить аспект гео-объекту:

Пример запроса:

{

"geoObjectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"AspectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

### GeoObjectsClassifiers

GET /api/GOC/GetGeoObjectsClassifiers – получение всех связей типа гео-объект-классификатор:

Пример результата запроса:

[

{

"geoObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa",

"classifierId": "2676dc59-5489-4051-ba55-d14658b58127"

},

{

"geoObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa",

"classifierId": "7aae5929-66dc-4bd6-9c54-05390f001a6c"

},

{

"geoObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa",

"classifierId": "9fc0590c-90bd-4749-94ae-ec981650e27e"

}

]

### ParentChild

POST /api/ParentChild/Link – добавление связи типа родитель-ребенок между двумя объектами:

Пример запроса:  
{

"parentGeographicalObjectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"childGeographicalObjectId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

GET /api/ParentChild/Link – получение связи типа родитель-ребенок:

Пример результата запроса:

{

"id": "0c527015-a686-4c2f-a824-68c611ba5360",

"parentGeographicalObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa",

"childGeographicalObjectId": "96bbdbcb-3ca4-4154-8ec5-fbbb9c0e7dbe"

}

DELETE /api/ParentChild/{id} – удаление связи типа родитель-ребенок:

### Topology

GET /api/Topology – получение всех топологических связей:

Пример результата запроса:

[

{

"id": "82a79750-e4fe-4df6-b1c0-928d656a5bfa",

"commonBorder": [59.95775871067945,30.400375014205686],[59.95940320281114,30.40732626404227],

"geographicalObjectInId": cdf21b84-591b-4795-8e32-aba646ce44aa,

"geographicalObjectOutId": 96bbdbcb-3ca4-4154-8ec5-fbbb9c0e7dbe

}

]

POST /api/Topology – добавление топологической связи между двумя объектами:  
 Пример запроса:

{

"geographicalObjectInId": "96bbdbcb-3ca4-4154-8ec5-fbbb9c0e7dbe",

"geographicalObjectOutId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"

}

DELETE /api/Topology/{id} – удаление топологической связи между двумя объектами.

### Aspect

GET /api/Aspects – получение всех аспектов:  
 Пример результата запроса:

[

{

"id": "8ccbe92b-ed0f-499c-847a-61b384ce88bb",

"type": "Aspect\_1",

"geographicalObjectId": "cdf21b84-591b-4795-8e32-aba646ce44aa"

}

]

POST /api/Aspects – создание аспекта:  
 Пример запроса:  
 {

"type": "Aspect\_1",

"geographicalObjectId": " cdf21b84-591b-4795-8e32-aba646ce44aa "

}

GET /api/Aspects/{id} – получение аспекта по его Id.

DELETE /api/Aspects/{id} – удаление аспекта.

GET /api/Aspects/CallAspect

### Classifier

GET /api/Classifier – получение всех классификаторов:

Пример результата запроса:  
 [

{

"id": "7aae5929-66dc-4bd6-9c54-05390f001a6c",

"name": "class3",

"code": null,

"commonInfo": null

},

{

"id": "2676dc59-5489-4051-ba55-d14658b58127",

"name": "class2",

"code": null,

"commonInfo": null

},

{

"id": "9fc0590c-90bd-4749-94ae-ec981650e27e",

"name": "class1",

"code": null,

"commonInfo": null

}

]

GET /api/Classifier/{id} – получение классификатора по его Id.

POST /api/Classifier – создание классификатора:

Пример запроса:  
 {

"id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",

"name": "string",

"code": "string",

"commonInfo": "string"

}

DELETE /api/Classifier/{id} – удаление классификатора по его Id.

# Работа с клиентской частью

Клиентская часть системы взаимодействует с сервером через REST API. Взаимодействие включает следующие шаги:

1. **Отправка запросов**: Клиент отправляет запросы к серверу для создания, изменения, удаления и получения информации о географических объектах и их связях.
2. **Получение ответов**: Сервер обрабатывает запросы и возвращает ответы с соответствующими данными.
3. **Обработка данных**: Клиентская часть обрабатывает полученные данные и отображает их пользователю.

## Пример использования API в клиентском приложении

Клиентская часть может использовать API для выполнения различных операций. Пример использования API для создания нового объекта:

fetch('/api/geoobjects', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({

name: 'Новый объект',

geoNameId: 123,

status: 'Активен',

geometry: {

type: 'Point',

coordinates: [30.0, 50.0]

}

})

})

.then(response => response.json())

.then(data => {

console.log('Созданный объект:', data);

})

.catch(error => {

console.error('Ошибка:', error);

});

# Документация и поддержка

**Описание документации API**

Документация API предоставляется с помощью Swagger. Swagger генерирует интерактивную документацию, которая позволяет легко просматривать доступные конечные точки, их параметры и примеры запросов и ответов.

**Поддержка и расширение системы**

Система спроектирована таким образом, чтобы ее можно было легко расширять и модифицировать. Поддержка новых типов геосистем и связей может быть реализована путем добавления новых моделей и конечных точек API.

# Заключение

В ходе разработки серверной части системы многоаспектного геоинформационного моделирования были достигнуты следующие результаты:

* Разработана и протестирована REST API для создания, управления и поиска географических объектов и их связей.
* Обеспечена поддержка топологических и семантических отношений между объектами.
* Реализована возможность расширяемой поддержки классификаторов геосистем.
* Система успешно взаимодействует с клиентской частью, предоставляя необходимые данные для визуализации и управления объектами.

Система предоставляет надежную и гибкую платформу для дальнейшего развития и интеграции с другими компонентами геоинформационных систем.

# Список использованной литературы

* 1. metanit.com/sharp/efcore (руководство по Entity Framework Core 8)
  2. learn.microsoft.com/ru-ru/ef (центр документации Entity Framework)
  3. swagger.io/docs (документация по Swagger)
  4. blazor.syncfusion.com/documentation/maps/populate-data (заполнение данных в компоненте Blazor Maps)
  5. www.postgresql.org/docs (документация по PostgresSQL)
  6. git-scm.com/doc (документация по git)
  7. Уолтер Шилдс – SQL: быстрое погружение
  8. Марк Прайс - C# 10 и .NET 6. Современная кроссплатформенная разработка (2023)
  9. github.com/dotnet/AspNetCore.Docs (документация по ASP NET CORE)