Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем
Направление подготовки 09.04.04 – Программная инженерия
Направленность (профиль) образовательной программы Управление разработкой программного обеспечения

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой
_____ А.В. Бушманов
«____»_____2021 г.

МАГИСТЕРСКАЯ РАБОТА

на тему: Разработка приложения для обработки ГНСС-данных с применением геоинформационных технологий

Исполнитель
студент группы 957-ом _____ (подпись, дата) А.В. Понизов

Руководитель
доцент, канд. техн. наук _____ (подпись, дата) Т.А. Галаган

Руководитель научного содержания программы магистратуры
профессор, д-р техн. наук _____ (подпись, дата) И.Е. Еремин

Нормоконтроль
инженер кафедры _____ (подпись, дата) В.Н. Адаменко

Рецензент _____ (подпись, дата) О.Г. Какаулин

Благовещенск 2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

Факультет математики и информатики
Кафедра информационных и управляющих систем

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Бушманов
«____» _____ 2021 г,

З А Д А Н И Е

К магистерской диссертации студента Понизова Александра Викторовича

1. Разработка приложения для обработки ГНСС-данных с применением геоинформационных технологий.

(утверждена приказом от 01.03.2021 №412-уч)

2. Срок сдачи студентом законченной работы (проекта): 23 июня 2021 г.

3. Исходные данные к магистерской диссертации: отчет о прохождении преддипломной практики, ГОСТы, научные публикации, дополнительная литература.

4. Содержание магистерской диссертации (перечень подлежащих разработке вопросов): обоснование необходимости разработки программного продукта, сравнительный анализ микросервисной архитектуры, описание алгоритмического и программного обеспечения при разработке приложения для обработки ГНСС-данных, разработка и тестирование приложения.

5. Дата выдачи задания: 25 февраля 2021г.

Руководитель магистерской диссертации: Галаган Т.А., доцент, кандидат технических наук.

Задание принял к исполнению: _____

РЕФЕРАТ

Магистерская диссертация содержит 91 с., 61 рисунков, 5 таблиц, 39 источников.

ВЕБ-ФРЕЙМВОРКИ, ПРИЛОЖЕНИЕ, ГЛОБАЛЬНЫЕ НАВИГАЦИОННЫЕ СПУТНИКОВЫЕ СИСТЕМЫ, ГНСС-ДАННЫЕ, ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ДАННЫХ, ТЕСТИРОВАНИЕ, .NET CORE, C#

Цель исследования: разработка приложения, основанного на микросервисной архитектуре, для автоматизации процесса сбора, систематизации, конвертирования и обработки данных, получаемых от глобальных навигационных спутниковых систем (ГНСС).

Научная новизна исследования заключается в использовании совокупности технологий (ASP NET CORE и BLAZOR) для создания приложения оптимизирующего процессы, связанных с данными, получаемыми в результате применения метода космической геодезии. Данные технологии ранее не использовались для решения подобных задач, их применение позволит уменьшить время затрачиваемое на обработку таких данных и как следствие время на получение научных результатов, что, в свою очередь, поможет специалистам оперативно определять статистические и количественные характеристики полей напряженно-деформированного состояния разломно-блоковых структур северо-восточной части Амурской литосферной плиты и зон её взаимодействий с Евразийской и Охотской литосферными плитами.

Предмет исследования: подходы разработки современного приложения для обработки ГНСС-данных с применением геоинформационных технологий.

Выполнение диссертации включает три этапа:

Первым этапом является произведение анализа предметной области. В результате было рассмотрено применение методов космической геодезии в задачах анализа геодинамической активности, произведен сравнительный анализ

микросервисной архитектуры с монолитной и сервисноориентированой архитектурами.

На втором этапе выполнен анализ взаимодействия пользователя и приложения, алгоритмов работы с входящими данными, определены функции приложения, произведено проектирование взаимодействие модулей приложения. Также были сформированы требования к программному продукту.

На третьем этапе обоснован выбор средств разработки приложения, произведена характеристика выбранного програмно-технического обеспечения. Затем описаны основные этапы разработки программного продукта, произведен анализ достоверности и практической значимости полученных результатов.

В результате разработано и запущено в тестовом режиме приложение обработки ГНСС-данных. Основанное на микросервисной архитектуре, оно покроет потребность в обработке ГНСС-данных и обеспечит их полный жизненный цикл: регулярный сбор, систематизацию, хранение и обработку. Создание приложения позволило сократить время обработки двадцати дневного проекта (данные с пяти станций за период в двадцать дней) до двух часов.

СОДЕРЖАНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

В настоящей магистерской диссертации использованы ссылки на следующие стандарты и нормативные документы:

ГОСТ 19.001-77 ЕСПД Общие положения

ГОСТ 19.002-80 ЕСПД Схемы алгоритмов и программ. Правила выполнения

ГОСТ 19.003-80 ЕСПД Схемы алгоритмов и программ. Обозначения условные графические

ГОСТ 19.004-80 ЕСПД Термины и определения

ГОСТ 19.101-77 ЕСПД Виды программ и программных документов

ГОСТ 19.102-77 ЕСПД Стадии разработки

ГОСТ 19.103-77 ЕСПД Обозначения программ и программных документов

ГОСТ 19.104-78 ЕСПД Основные надписи

ГОСТ 19.105-78 ЕСПД Общие требования к программным документам

ГОСТ 19.106-78 ЕСПД Требования к программным документам, выполненным печатным способом

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

ЯП – язык программирования;

ГНСС – глобальные навигационные спутниковые системы;

ГНСС-данные – данные получаемые от глобальных навигационных спутниковых систем;

API – Application Programming Interface – программный интерфейс приложения;

CQRS – Command Query Responsibility Principe – принцип разделения операций на команды и запросы;

DDD – Domain-Driven Design – предметно-ориентированное проектирование;

MSA – Microservice Architecture – Микросервисная архитектура;

Onion Architecture – Луковичная архитектура;

Pipeline – сформированная цепочка неких операций или действий, применяемая к входным данным;

REST – Representational State Transfer – передача состояния представления;

Rinex – открытый формат ГНСС-данных;

SOA – Service-oriented architecture – Сервис ориентированная архитектура;

HTTP – hypertext transfer protocol – протокол передачи гипертекста;

HTTPS – hypertext transfer protocol secure – безопасный протокол передачи гипертекста.

ВВЕДЕНИЕ

В соответствии с представлениями глобальной тектоники плит, разделяемыми большинством ученых, территория Верхнего Приамурья относится к стабильной части Евразийской литосферной плиты и образована в результате закрытия Монголо-Охотской океанической области путем смыкания Северо-Азиатского кратона и Амурского микроконтинента [1]. Однако, великое японское землетрясение 11 марта 2011 г. Mw=9.0 и целая серия последующих землетрясений вдоль границ Амурской литосферной плиты ярко продемонстрировало необходимость и актуальность изучения современных геодинамических процессов данной тектонической единицы.

В настоящее время, для исследования движений и деформаций как локального, так и регионального и глобального масштабов, причем как природного, так и техногенного характера используются спутниковые технологии. Уровень их точности достаточен для фиксации деформаций в пределах внутриконтинентальных районов Азии, как показано в работах [2, 3].

Сбор данных на исследуемой территории и их последующая обработка производится ручным способом, поэтому разработанное приложение автоматизирует процессы сбора и систематизации данных получаемых от спутников, их математической обработки и последующей визуализации результатов, определяющих качество обработанных данных.

Цель исследования: разработка приложения, основанного на микросервисной архитектуре, для автоматизации процесса сбора, систематизации, конвертирования и обработки данных, получаемых от глобальных навигационных спутниковых систем (ГНСС).

Научная новизна исследования заключается в применении совокупности технологий (ASP NET CORE и BLAZOR) для создания приложения оптимизирующего процессы, связанных с данными, получаемыми в результате применения метода космической геодезии. Данные технологии ранее не использовались для решения подобных задач, их применение позволит уменьшить вре-

мя затрачиваемое на обработку таких данных и как следствие время на получение научных результатов, что, в свою очередь, поможет специалистам оперативно определять статистические и количественные характеристики полей напряженно-деформированного состояния разломно-блоковых структур северо-восточной части Амурской литосферной плиты и зон её взаимодействий с Евразийской и Охотской литосферными плитами.

Предмет исследования: подходы разработки современного приложения для обработки ГНСС-данных с применением геоинформационных технологий.

Для достижения цели работы, необходимо выполнение следующих задач:

а) разработка программного модуля сбора ГНСС-данных

б) разработка программного модуля конвертирования ГНСС-данных в RINEX формат

в) разработка программного модуля по сбору, хранению и систематизации геодезических данных (ГНСС-данных), получаемых от ГНСС-станций;

г) разработка программного модуля обработки ГНСС-данных;

д) создание приложения обработки ГНСС-данных, состоящего из микросервисов, использующих программные модули и взаимодействующих друг с другом по протоколу HTTP на основании строго регламентированного API, включающего в себя:

1) микросервис сбора, индексации и конвертирования данных;

2) микросервис обработки геодезических данных, так же реализующий функционал: хранение результатов поэтапных расчетов различных величин; расчет статистической информации, основанной на вычисленных данных, которая позволит судить о характере геодинамической активности, а также фильтровать/маркировать станции с высоким уровнем зашумленности; отображение полученных результатов в виде 2d рисунков, в формате ghostscript;

3) микросервис графического интерфейса пользователя, построенного на фреймворке Blazor для обеспечения человеко-машинного взаимодействия.

Практическая значимость заключается в создании решения, внедрение которого позволит автоматизировать процессы сбора, хранения, конвертирования,

а также обработки информации. Использование микросервисной архитектуры, позволит обеспечить возможность горизонтальной масштабируемости системы, что в свою очередь позволит производить больше расчетов на единицу времени, а также увеличит гибкость в работе, уменьшит временную цену ошибочных расчетов и повысит общую продуктивность процесса. Приложение должно иметь адаптивный и современный Веб-интерфейс, базироваться на микросервисной архитектуре, и обеспечивать возможность гибкой настройки процесса анализа геодинамической активности.

В результате внедрения геоинформационной системы ожидается увеличение продуктивности и как следствие объема обрабатываемых данных, а также повышение надежности и отказоустойчивости в области хранения данных.

Исследования полей современных движений и напряжено-деформированного состояния земной коры актуальны также для решения прикладных геологических и инженерных задач. Особенно это важно в связи с развитием на юге Дальнего Востока России линейных и площадных объектов всероссийского и мирового значения (заводы, космодромы) [5].

Новизна и значимость технических решений подтверждена публикациями в научных изданиях.

Апробация результатов диссертационного исследования отражается в следующих научных публикациях:

1 Понизов А.В., Серов М.А., Галаган Т.А. Разработка Веб-приложения для обработки ГНСС-данных с использованием микросервисной архитектуры // Вестник АмГУ 2020. № 89. С. 27-31.

2 Понизов А.В., Галаган Т.А. Проектирование веб-приложения для обработки ГНСС-данных с использованием микросервисной архитектуры. //. Материалы 21 региональной научно-практической конференции «Молодежь 21 века: Шаг в бущее», Благовещенск, 2020, №4, С. 132-133.

3 Понизов А.В., Галаган Т.А. Преимущества архитектурного CQRS-подхода в высоконагруженных приложениях. // Материалы 14 международной научной конференции «САМ 2020», Благовещенск, 2020, С.81-84.

4 Понизов А.В., Галаган Т.А. Преимущества микросервисной архитектуры при разработке современных приложений // «День науки»: материалы XXIX научной конференции Амурского государственного университета (23-25 ноября 2020 г., Благовещенск), С.37-38.

Так же результаты диссертационного исследования были представлены на научной конференции:

1 Понизов А.В., Галаган Т.А. Преимущества микросервисной архитектуры при разработке современных приложений // День науки АмГУ (Благовещенск 2020)

1 ОСОБЕННОСТИ СОВРЕМЕННЫХ МЕТОДОВ ОБРАБОТКИ ГНСС-ДАННЫХ

1.1 Актуальность задачи обработки ГНСС-данных

Регион для исследования которого применяется метод космической геодезии и как следствие необходимо собирать и обрабатывать данные, находится в пределах тройного сочленения Центрально-Азиатского складчатого пояса (ЦАСП), Сибирской платформы и Тихоокеанского складчатого пояса. На более детальном уровне он отражает взаимодействие тектонических единиц Аргунского континентального массива, Монголо-Охотского складчатого пояса (МОСП), Селенга-Станового и Джугджуро-Станового блоков [6].

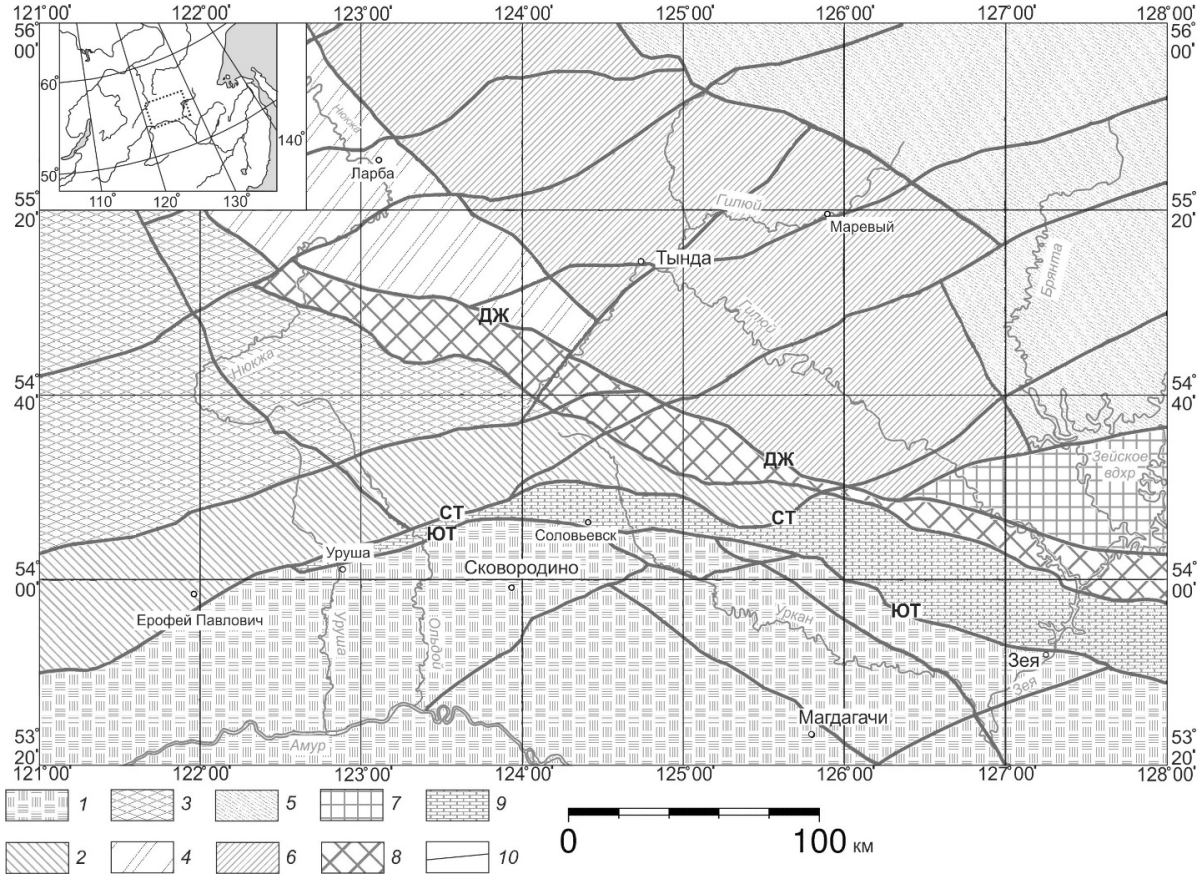

Рисунок 1 – Карта структуры земной поверхности части исследуемого региона

Данные за более чем 100-летную историю сейсмических наблюдений позволяют утверждать, что на выделяемой территории современный этап геодинамической эволюции контролируется транспрессионым (сжатие со скольжением)

сближением Амурской и Евразийской плит.

Следует отметить тот факт, что западная часть территории верхнего Приамурья сочленена с Байкальской рифтовой системой – одной из самых тектонически активных внутриконтинентальных зон планеты, для которой характерными признаками являются расчлененный рельеф, высокая сейсмичность, масштабное проявление кайнозойского вулканизма, развитая разломная сеть.

Анализ сейсмических и гравиметрических данных, позволяет сделать заключение о наличие в пределах исследуемого региона высокоамплитудных сдвиговых перемещений, в том числе и происходивших на неотектоническом этапе, осложненных широкомасштабным распространением надвигов [7]. Что также подтверждается другими геологическими исследованиями и данными бурения. Многие авторы отмечают широкое развитие в земной коре наклонных границ, ассоциируемых ими с разломами, причем по характеру изменения плотности в слоях выделяются как разломы сжатия, выражающиеся во встречном нарастании плотности, так и разломы растяжения, с обратной картиной распределения плотности.

1.2 Глобальные навигационные спутниковые сети как метод измерения геодинамической активности

Одним из наиболее эффективных методов анализа геодинамической активности является космическая геодезия с применением глобальных навигационных спутниковых систем (ГНСС) [8]. Этот метод позволяет исследовать движения и деформации как локального, регионального, так и глобального масштаба. Точность таких измерений – до 0,1 сантиметра – достаточна для фиксации деформаций в пределах внутриконтинентальных районов.

Первыми крупными землетрясениями, изученными с использованием ГНСС стали землетрясения 1987 и 1989 годов в Калифорнии, с магнитудами 6.6 и 7.1 соответственно [9].

К мировым примерам косейсмических смещений, зарегистрированных этим методом, можно отнести:

- кашмирское землетрясение в Индии 8 октября 2005 года M7.6 – ам-

плитуда смещений 26 см;

- олюторское землетрясение на Камчатке 20 апреля 2006 года М7.6 – максимальное горизонтальное смещение – 3,2 м;

- землетрясение Тохоку в Японии 11 марта 2011 года М9,0 – максимальное горизонтальное смещение – 4 м, вертикальное – 0,6 м.

ГНСС-метод может использоваться в двух режимах [10]:

- непрерывный режим, когда длительные наблюдения выполняются на постоянно действующих станциях, что позволяет достигать значения самой высокой точности и улавливать кратковременные деформации;

- режиме полевых кампаний, в ходе которых периодически повторяющиеся кратковременные измерения проводятся на реперных пунктах геодезической сети.

Для непрерывного режима необходимо отметить большое количество создаваемой приемниками информации в день (порядка 10 Гб. в день), и её высокой распределенности. Это обуславливается двумя параметрами измерений: частотой дискретизации сигнала (1 сек.) при обмене информацией между принимающей станцией и спутниками, и количеством станций-приемников, расположенных на исследуемой территории.

ГНСС-метод использует три сегмента [11]:

- космический сегмент – состоит из групп навигационных спутников, выполняющих роль формирователей и излучателей сигналов, необходимых для навигационных определений потребителей;

- наземный сегмент – в его состав входят космодром, командно-измерительный комплекс и центр управления. Космодром обеспечивает вывод спутников на требуемые орбиты при первоначальном развертывании навигационной системы, а также периодическое восполнение спутников по мере их выхода из строя или выработки ресурса;

- пользовательский сегмент, используемый в рамках задачи, состоит из базовых ГНСС-станций, для которых и происходит определение координат.

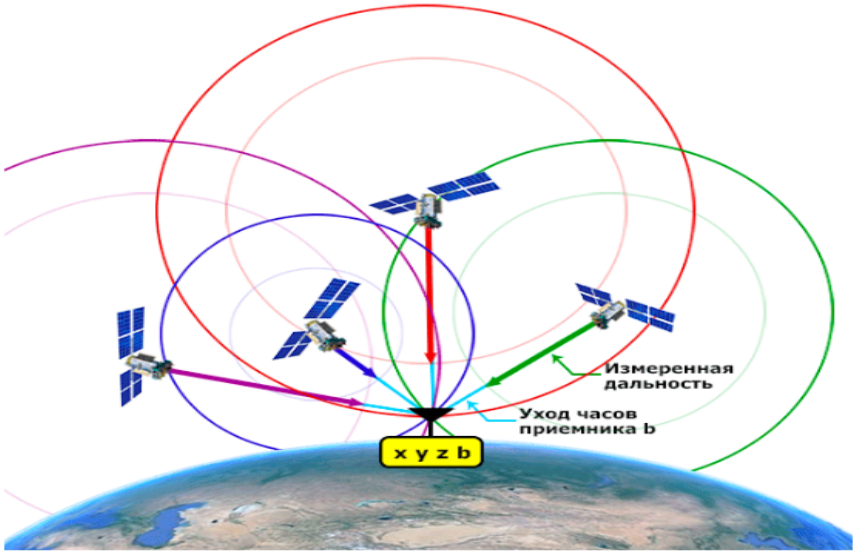

Рисунок 2 – Позиционирование посредством GPS

Процесс, в рамках подхода космической геодезии, состоит из нескольких этапов:

- сбор «сырых» данных на исследуемой территории, посредством использования стационарных и портативных ГНСС-приемников;
- загрузка «сырых» данных с приемников на локальные хранилища;
- преобразование «сырых» данных в открытый формат RINEX;
- загрузка файлов спутниковых орбит, навигационных и дополнительных файлов, необходимых для последующих этапов;
- обработка данных с помощью различных математических методов;
- проверка предварительных результатов;
- получение итоговых результатов и их визуализация.

Современная спутниковая навигация основывается на использовании принципа без запросных дальномерных измерений между навигационными спутниками и потребителем. Это означает, что потребителю передается в составе навигационного сигнала информация о координатах спутников. Синхронно производятся измерения дальностей до навигационных спутников.

Способ измерений дальностей основывается на вычислении временных задержек принимаемого сигнала от спутника по сравнению с сигналом, генери-

руемым аппаратурой потребителя.

1.3 Структура данных получаемых от глобальных навигационных спутниковых систем

Работа приложения начинается с загрузки ГНСС-файлов, являющихся результатом работы базовой станции являются содержащие внутри служебную информацию о спутниках в зоне видимости, а также навигационные сообщения [12] и следующие типы наблюдений: L1 и L2 – фазовые измерения на несущих частотах (в циклах), C1 – значение псевдодальности вычисленное по C/A коду (в метрах), P1 и P2 – значение псевдодальности вычисленное по P-коду двух несущих частот (в метрах), S1 и S2 – значение сигнал/шум для двух несущих частот.

- эфемеридная информация, необходимая для вычисления координат спутника с достаточной точностью;
- погрешность расхождения бортовой шкалы времени относительно системной шкалы времени для учета смещения времени космического аппарата при навигационных измерениях;
- расхождение между шкалой времени навигационной системы и национальной шкалой времени, для решения задачи синхронизации потребителей;
- признаки пригодности с информацией о состоянии спутника для оперативного исключения спутников с выявленными отказами из навигационного решения;
- альманах с информацией об орбитах и состоянии всех аппаратов в группировке для долгосрочного грубого прогноза движения спутников и планирования измерений;
- параметры модели ионосферы, необходимые одночастотным приемникам для компенсации погрешностей навигационных измерений, связанных с задержкой распространения сигналов в ионосфере;
- параметры вращения Земли для точного пересчета координат потребителя в разных системах координат.

1.4 Сравнительный анализ существующих программных решений

В рамках поставленной цели отсутствуют программные продукты, обеспечивающие построение автоматизированного процесса.

Существуют программные продукты для решения отдельных задач в рамках ГНСС-метода. Однако, именно отсутствие комплексных продуктов затрудняет работу специалистов, поскольку различные программные продукты, написанные на разных языках, имеют различные интерфейсы и форматы данных, а также поддерживаются различными коллективами разработчиков. Это часто приводит к невозможности использовать эти продукты для сложной обработки данных и/или применять один программный продукт для дальнейшей обработки результатов, полученных в другом продукте.

Для ускорения и сохранения передовых стандартов качества получаемых результатов, в модуле обработки информации будет применяться математический пакет GAMIT/GLOBK поставляемый Массачусетским технологическим институтом. Данный пакет прикладных программ широко распространен в прикладной области, однако имеет ряд существенных недостатков, а именно: отсутствие масштабируемости; низкая производительность; несоответствие стандартам современных консольных linux приложений. Данные отрицательные качества являются следствие немаленького возраста продукта (создан в 1989 году) и как следствие использование устаревших концепций и технологий. И несмотря на ежеквартальный цикл обновления программного продукта, кардинальных изменений не происходит.

Создание программного интерфейса над данным пакетом, с использованием высокоуровнего языка C#, позволит обеспечить высокую степень параллелизма выполняемых задач, высокоуровневый объектно-ориентированный интерфейс и строгую типизацию входных параметров. А создание на этой основе микросервиса, позволит обеспечить веб доступ, простоту развертывания и высокую горизонтальную масштабируемость.

Создание математического пакета с нуля является ресурсоемким и затратным процессом и, несмотря на очевидные плюсы, потребует больших финансо-

вых вложений, что в рамках поставленной цели является излишним. Построение высокоуровневой программной обертки на языке C# обеспечит те же преимущества, но с более низкой временной и денежной стоимостью.

Для первичной обработки данных специалистами используется проприетарное программное обеспечение фирмы поставщика базовых станций. Однако его использование в рамках поставленной цели невозможно, во-первых, из-за лицензии, во-вторых, из-за закрытого кода приложения. Так же, существенным недостатком является отсутствие интерфейса командной строки.

Следовательно, в рамках постеленной цели будет применена открытая пользовательская библиотека runpkr, позволяющая выполнить преобразование из проприетарного двоичного формата ГНСС-данных в открытый двоичный формат dat. Однако, необходимо будет реализовать процедуру преобразования из dat в Rinex (общепринятый открытый формат ГНСС-данных).

1.5 Сравнительный анализ микросервисной архитектуры

Для уменьшения времени, требуемого на произведение расчетов, обеспечения отказоустойчивости и горизонтальной масштабируемости было принято решение разрабатывать приложение, основываясь на микросервисной архитектуре.

Микросервисная архитектура [13] – это подход к созданию приложения, подразумевающий отказ от единой, монолитной структуры. То есть вместо того, чтобы исполнять все ограниченные контексты приложения на сервере с помощью внутрипроцессных взаимодействий, мы используем несколько небольших приложений, каждое из которых соответствует какому-то ограниченному контексту. Причём эти приложения работают на разных серверах и взаимодействуют друг с другом по сети, например посредством HTTP.

Планируемая схема приложения обработки ГНСС-данных представлена на рисунке 3. На схеме изображено приложение, состоящее из контейнеров. Контейнер – это запущенный образ микросервиса. Образ – это приложение, модули и зависимости от операционной системы необходимые для его работы. В рамках работы планируется создание контейнера приложения выполняющего

процесс сбора, систематизации и хранения данных, контейнера для приложения обработки данных и контейнера для подсистемы хранения данных (представленной документориентированной системой хранения MongoDB и реляционной базой данной MySQL.

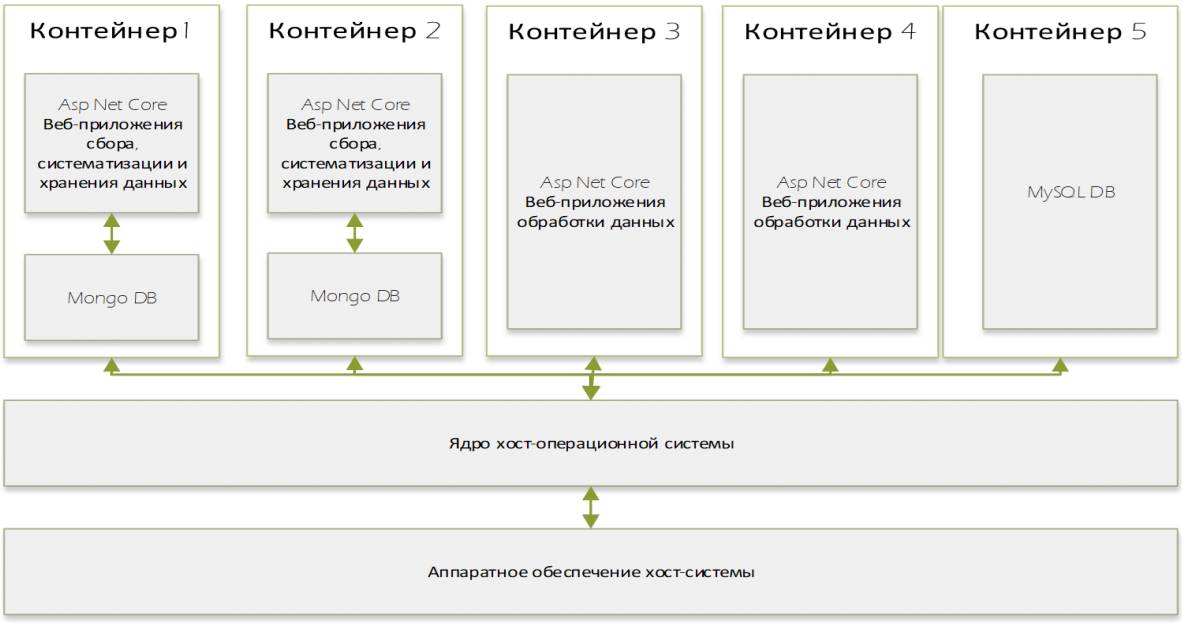

Рисунок 3 – Схема контейнеризации сервисов

Сервис-ориентированная архитектура (рисунок 4) предусматривает модульность разработки и слабую связанность компонентов, поэтому получаем изолированную и распределённую систему.

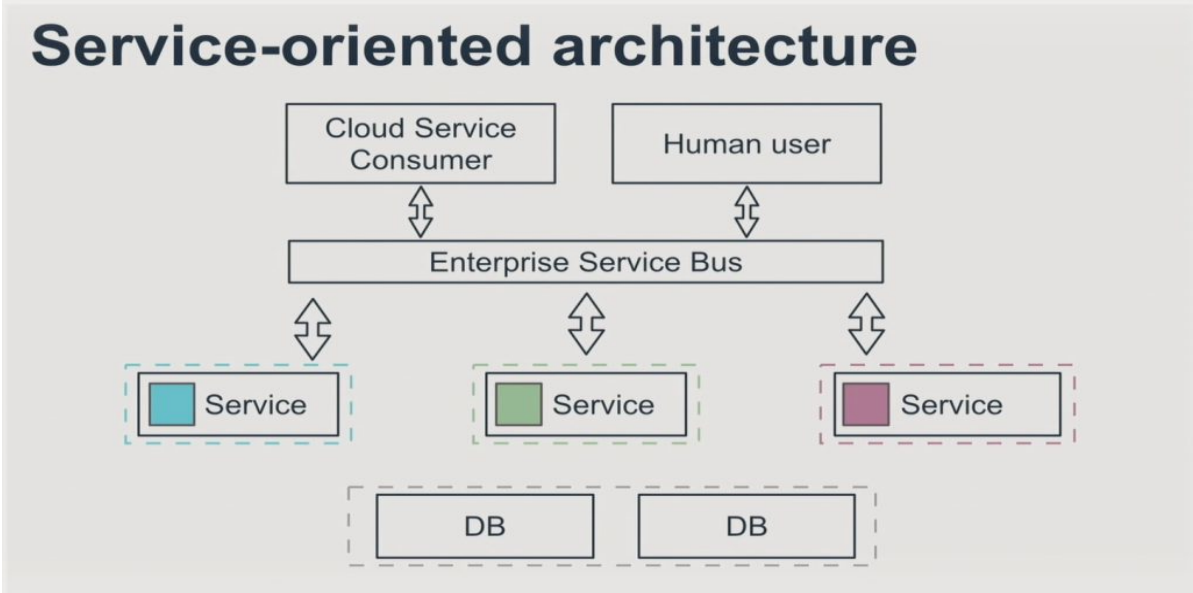

Рисунок 4 – Сервис-ориентированная архитектура

Однако современная концепция микросервисной архитектуры, предполагает некоторое изменение постулатов сервис-ориентированной архитектуры (рисунок 5)

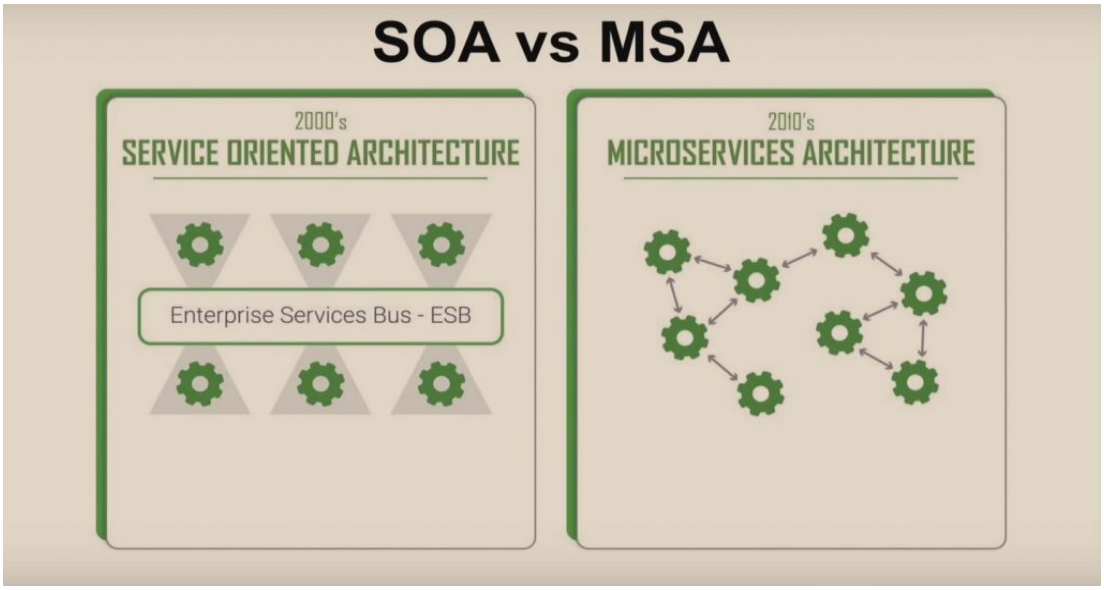

Рисунок 5 – Сравнение схем взаимодействия компонент внутри сервис-ориентированной и микросервисной архитектур

Основные аппаратные и программные преимущества при использовании микросервисов приведены в таблице 1.

Таблица 1 – Преимущества микросервисов

Микросервисы	
Аппаратные преимущества	Программные преимущества
Независимая масштабируемость Модули могут быть масштабированы при размещении на разных узлах	Сохранение модульности: Микросервисы инкапсулируют приложение и его зависимости, в том числе от операционной системы. Так же микросервисы физически изолированы друг от друга, имеют собственную файловую и сетевую подсистемы.
Независимый технический стек Так как микросервисы изолированы физически и взаимодействуют посредством сетевых протоколов (например HTTP) они могут быть написаны на различных языках программирования	Независимая эволюция подсистем Могут работать несколько версий микросервиса одновременно. Старая версия будет поддерживать обратную совместимость необходимое время, а микросервис с новой версией внедрит новые функции.

Основные различия микросервисной архитектуры и классической [5] (монолитной) приведены в таблице 2.

Таблица 2 – Сравнение монолитной и микросервисной архитектуры

Монолитная архитектура	Микросервисы
Простота Монолитная архитектура гораздо проще в реализации, управлении и развёртывании. Микросервисы требуют тщательного управления, поскольку они развёртываются на разных серверах и используют API.	Частичное развёртывание Микросервисы позволяют по мере необходимости обновлять приложение по частям. При единой архитектуре нам приходится заново развёртывать приложение целиком, что влечёт за собой куда больше рисков.
Согласованность (Consistency) При монолитной архитектуре проще поддерживать согласованность кода, обрабатывать ошибки и т. д. Зато микросервисы могут полностью управляться разными командами с соблюдением разных стандартов.	Доступность У микросервисов доступность выше: даже если один из них сбоит, это не приводит к сбою всего приложения.
Межмодульный рефакторинг Единая архитектура облегчает работу в ситуациях, когда несколько модулей должны взаимодействовать между собой или когда мы хотим переместить классы из одного модуля в другой. В случае с микросервисами мы должны очень чётко определять границы модулей!	Сохранение модульности Сохранять модульность и инкапсуляцию может быть непросто, несмотря на правила SOLID. Однако микросервисы позволяют гарантировать отсутствие общих состояний (shared state) между модулями.
	Мультиплатформенность/гетерогенность Микросервисы позволяют использовать разные технологии и языки, в соответствии с вашими задачами.

К преимуществам микросервисной архитектуры можно отнести: частичное развертывание, доступность, переносимость и мультиплатформенность.

1.6 Недостатки использования микросервисной архитектуры

Необходимость управлять гибкостью технологии

Одно из преимуществ микросервисов заключается в том, что мы можем применять разные технологии для решения одной и той же задачи. Например, в каждом микросервисе использовать разные библиотеки для XML-парсинга или

разные инструменты сохранности данных. Но сама возможность не означает, что мы должны это делать. Не исключено, что обилие технологий и библиотек выйдет из-под контроля. Так что выберите базовый набор инструментов и обращайтесь к другим только тогда, когда это действительно нужно.

Необходимость управлять нестабильностью интерфейса

В начале разработки микросервиса его API особенно нестабилен. Но даже на более поздних стадиях, нам приходится менять API. Осторожно вносите изменения, потому что другие приложения будут полагаться на стабильность API.

Необходимо быть уверенными в согласованности данных

Микросервисы имеют собственные хранилища данных. И во многих случаях данные, принадлежащие одному микросервису, будут частично или целиком скопированы другим, клиентским микросервисом. Когда данные у поставщика меняются, он инициирует событие для запуска обновления данных, скопированных клиентским микросервисом. Событие попадает в очередь сообщений и ожидает, когда его получит клиентский микросервис.

Эта схема означает, что клиентский микросервис будет обладать устаревшими данными, пока не обнаружит нужное событие. Данные не согласованы. Конечно, в итоге изменения будут применены ко всем копиям, а данные снова станут согласованными. Это называется eventual consistency – «согласованность в конечном счёте». Этот эффект имеет важное значение в ходе разработки приложений, от серверной части до UX-уровней.

2 АЛГОРИТМИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ РЕШЕНИЯ ЗАДАЧИ

2.1 Анализ требований к микросервисам

2.1.1 Требования к внешним интерфейсам

Микросервисы должны предоставлять клиентам точки HTTP-соединений. Взаимодействуя с данными точками, другие компоненты системы смогут получить/отправить данные или выполнить какие-либо действия. Данные точки представляют из себя url адреса и могут отличаться у каждого микросервиса. Для того, чтобы взаимодействовать с данными точками клиенту необходимо отправить HTTP запрос: GET, POST, PUT, DELETE и данные на url адрес, соответствующий точке.

Микросервис систематизации, сбора и хранения ГНСС-данных

Таблица 3 – Основные функции API

Запрос	URI	Действие
POST	api/gpsfileinfo	Индексирует файлы
GET	api/gpsfileinfo?{dateRange}	Получает имена существующих файлов за выбранный период
GET	api/gpsfile/{id}	получает файл по id
GET	api/gpsfile/{name}	получает файл по имени

Микросервис обработки ГНСС-данных

Точки HTTP-соединений для микросервиса обраработки данных показаны в таблице 4.

Одной из проблем при разработке микросервисов, может стать необходимость изменения Url адреса внешнего интерфейса. Существует несколько вариантов решения данной проблемы, но самый простой, это разработать и зафиксировать внешний API исходя из спецификации и требований к конкретному микросервису, а потребность в изменениях удовлетворять за счет версионирования. То есть один микросервис может иметь несколько версий внешнего API, реализующих функционал.

Таблица 4 – Основные функции API

Запрос	URI	Действие
POST	api/project	создает проект
GET	api/project	получает информацию о проектах
POST	api/project/symlink	выполняет ковертирование и создает ссылки на файлы
GET	api/operationStatus/log/{projectName}	возвращает данные из буфера логов текущей задачи
GET	api/operationStatus/progress/{projectName}	возвращает прогресс текущей задачи в %
POST	api/gpsFileLoad	загружает файлы в проект
POST	api/sh/{scriptName}	

2.2 Определение основных функций, требуемых к реализации в программном обеспечении

Для создания спецификации алгоритмов, с учетом описанных выше требований, были выделены следующие функции:

- создание проекта (создание записи о проекте в базе данных, а именно название, начало и конец периода обработки, структура папок, результат обработки проекта);
- загрузка файлов необходимых для обработки;
- конвертирование файлов из проприетарного формата в Rinex формат;
- создание символических ссылок на загруженные файлы в директории проекта;
- запуск команд пакета Gamit/Globk;
- обеспечить логирование проводимых операций и доступ к логам;
- обеспечить доступ к прогрессу выполнения текущей задачи;
- наличие конфигурационного файла микросервиса и его изменение.

Функция – «создание проекта»

Отвечает за создание структуры проекта на жестком диске, записи в базе

данных и инициализации Gamit/Globk проекта.

Входная точка – HTTP POST запрос по пути «/api/project» принимает объект ProjectCreationCommand включающий в себя: имя проекта, имя пользователя, дату начала расчетов, дату окончания расчетов.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.Project.

Для обработки запроса обеспечивается последовательность операций:

- логирование входных данных;
- проверку входных данных;
- создание записи о проекте в СУБД;
- создание проекта в файловой системе;
- инициализация проекта начальными файлами;
- логирование результата выполнения запроса.

Функция – «загрузка файлов необходимых для обработки»

Загружает файлы необходимые для математической обработки проекта: орбиты спутников, навигационные файлы, ГНСС-файлы станций.

Входная точка – HTTP POST запрос по пути «/api/**gpsFileload**» принимает объект GpsFilesLoad включающий в себя: имя проекта, список имен загружаемых файлов.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.

Функция – «конвертирование файлов и создание символических ссылок»

Функция является дополнением к функции 1.2.2 и позволяет конвертировать файлы станций в Rinex-формат. Эта функция объединена с функцией создания символических ссылок на загруженные файлы.

Входная точка – HTTP POST запрос по пути «/api/**project/symlink**» принимает объект SymlinkCommand включающий в себя: имя проекта.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.

Функция – «запуск команд пакета Gamit/Globk»

Использование математического пакета Gamit/Globk для выполнения расчетов. Необходимо организовать логирование и доступ в реальном времени к логам для выполняемой задачи.

Входная точка – HTTP POST запрос по пути «/api/gamitGlobk/**{script name}**» принимает объект «[script name]Command» включающий в себя: имя проекта.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.

Функция – «логирование проводимых операций и доступ к логам»

Обеспечить логирование и доступ к логам задач математической обработки в реальном времени для отслеживания пользователями.

Входная точка – HTTP GET запрос по пути «/api/operationStatus/**log/{projectName}**» возвращает текущие данные в буфере лога операции проекта.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.

Функция – «обеспечить доступ к прогрессу выполнения текущей задачи»

Для задач, которые возможно измерить численно возвращать их прогресс в процентах.

Входная точка – HTTP GET запрос по пути «/api/operationStatus/**progress/{projectName}**» возвращает текущие данные в буфере лога операции проекта.

Логика работы функции должна располагаться в пространстве имен GIS.GamitGlobk.Application.Api.Features.

2.3 Брокер сообщений и микросервисное взаимодействие

Брокер сообщений позволяет приложениям подключаться к шине реализующей различные виды очередей. Благодаря этому, приложения, подключенные к брокеру, могут обмениваться сообщениями и совместно реализовывать функционал более крупного приложения. Обмен сообщениями является асинхронным, каналы связи между приложениями разделяются путем разделения отправляемых и получаемых данных по различным очередям. Количество по-

требителей подключенных к очереди не ограничено, так же как и не ограничено количество клиентов, что позволяет горизонтально масштабировать части приложения. На рисунке 7 изображена общая схема взаимодействия приложений с помощью брокера запросов.

В рамках приложения обработки ГНСС-данных брокер запросов планируется применять для масштабирования подсистем.

Рисунок 7 – Обмен сообщениями между приложениями с помощью брокера запросов

Это позволит создавать независимые структурные компоненты, менять и изменять их отдельно, несмотря на то что они совместно реализуют функционал «микросервиса».

Так приложение сбора, систематизации и хранения будет масштабировано для обеспечения стабильного доступа к данным, получаемым от ГНСС-станций. Масштабирование модуля (приложения) ответственного за обработку ГНСС-данных позволит пользователю обрабатывать столько проектов одновременно сколько необходимо.

Преимущества которые получит приложение при использовании брокера сообщений:

- асинхронный стиль взаимодействия (потребитель и производитель сообщений общаются асинхронно складывая сообщения в очереди, как результат ни один из участников не ожидает другого);
- репликация данных (потребитель может хранить копию данных до момента пока они изменятся, после чего получить обновленные данные как только они будут готовы).

2.4 Проектирование микросервисов

2.4.1 Анализ взаимодействия пользователя с приложением

Для формализации алгоритма-решения поставленных задач сформируем диаграммы: прецедентов, состояний, последовательностей взаимодействия пользователя и приложения. Диаграмма прецедентов показана на рисунке 9. На диаграмме изображен пользователь, который желает выполнить процедуру расчета тектонических сдвигов в выбранном регионе за какой-либо период времени. Для этого он выполняет вход в ГИС. После чего ему предоставляются следующие действия: удалить существующий проект, редактировать проект, просмотр результатов предыдущих проектов. Так же пользователь может запустить обработку проекта. В обработку проекта входят следующие действия:

- создание проекта обработки;
- выбор имени проекта и его размещения;
- выбор периоды обработки;
- запуск инициализации структуры проекта;
- обработка и получение результатов за выбранный период.

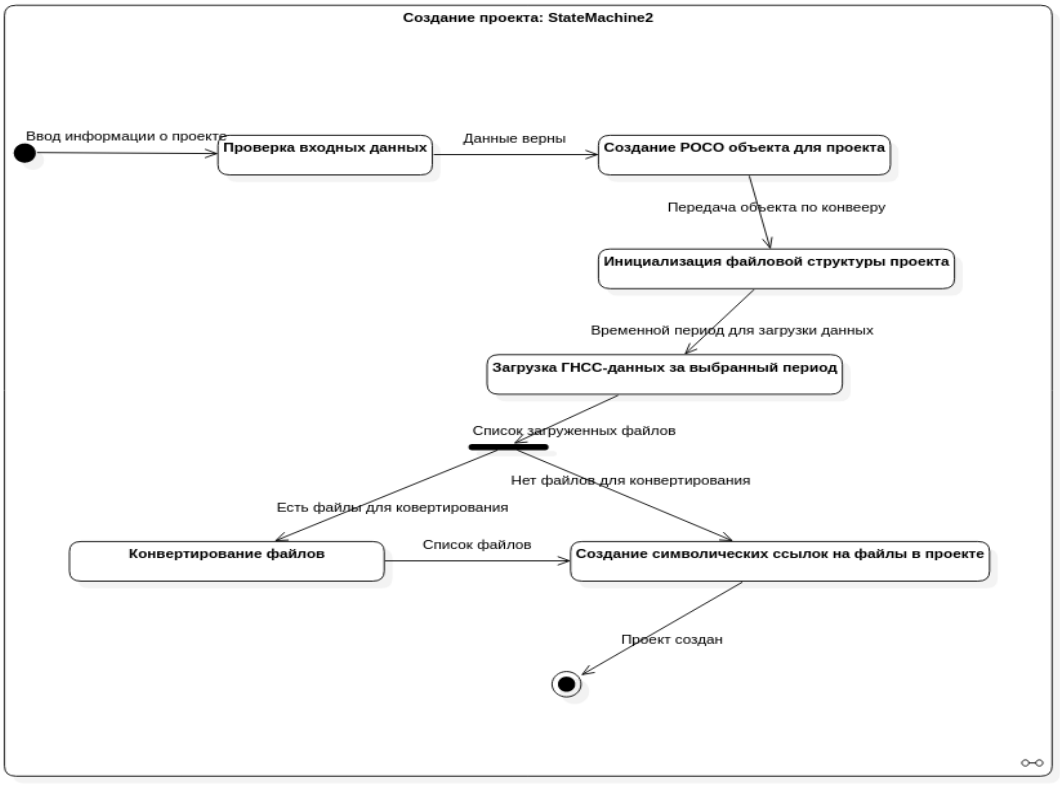

Рисунок 8 – Диаграмма состояний процедуры «Создание проекта»

На рисунке 8 отображена диаграмма состояний для процедуры создания проекта. Данная диаграмма отражает устройство конечного автомата, представляющего последовательность действий выполняемых микросервисом обработки для успешного создания проекта.

На вход микросервис принимает информацию о проекта (название, период обработки, расположение), затем он проверяет эти данные в соответствии с установленными правилами. Следующим является создание записи в базе данных (так как работка происходит через ORM, то ORM должна получить на вход объект класса, отражающего структуру таблицы). Затем микросервис создает структуру проекта в установленной директории файловой системы и выполняет загрузку ГНСС-файлов за установленный пользователем период из микросервиса сбора и хранения данных.

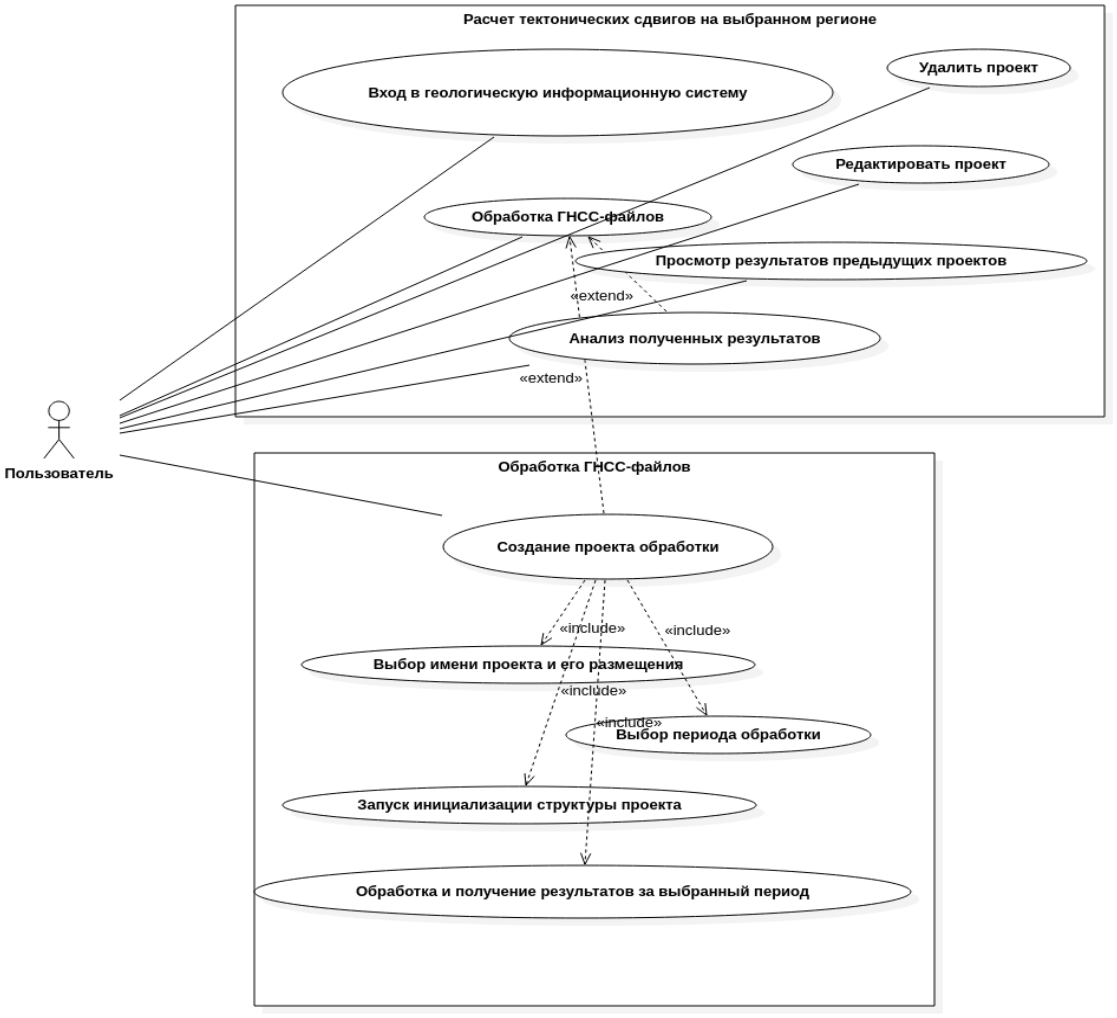

Рисунок 9 – Диаграмма прецедентов

Если среди файлов содержатся файлы в закрытом бинарном формате, то для них выполняется процедура конвертирования. Финальным действием является создание символических ссылок на файлы в директории проекта.

На рисунке 10 показана диаграмма состояний процедуры «Обработка проекта». Несмотря на то, что сама процедура является довольно сложной, её конечный автомат не велик.

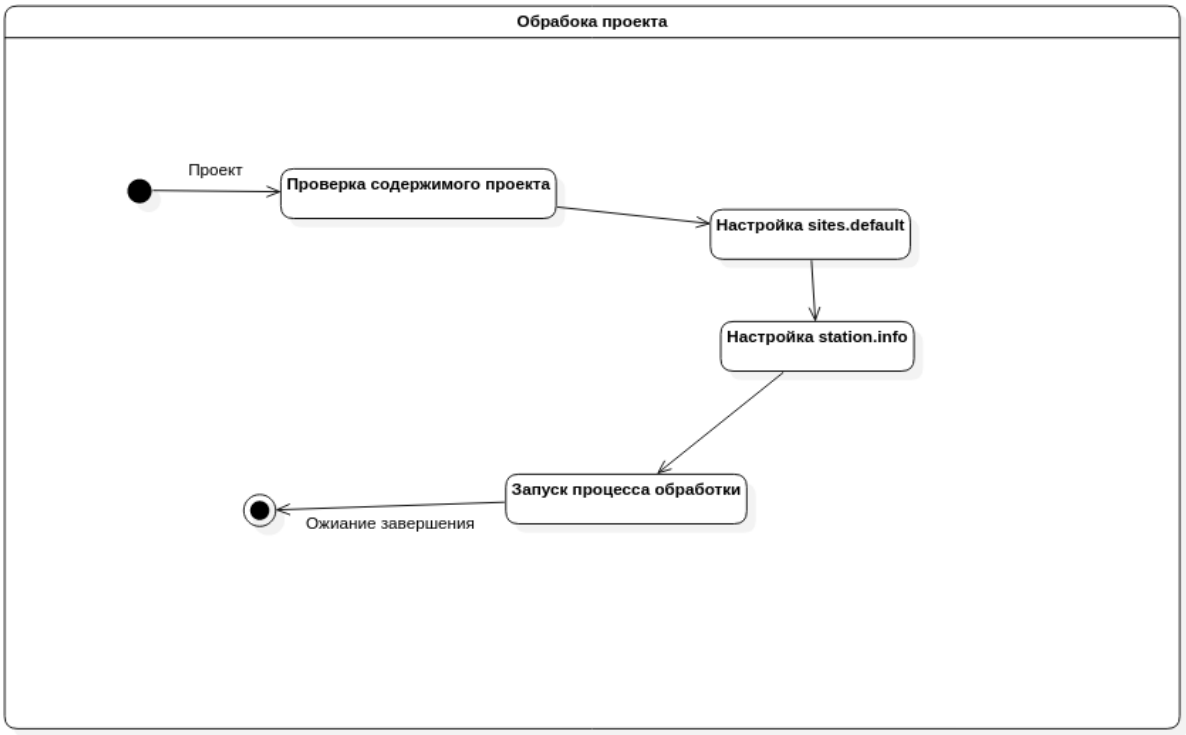

Рисунок 10 – Диаграмма состояний процедуры «Обработка проекта»

При обработки проекта происходит проверка его содержимого (наличие файлов для обработки, наличие подгруженных дополнительных файлов (файлы орбит, файлы траекторий спутников). Для обработки проекта микросервису потребуются заполненные пользователем конфигурационные файлы sites.default и station.info. В них пользователь настраивает проект (устанавливает контрольные точки и станции для них, выбирает производителя антенн).

После чего начнется запуск обработки проекта.

На рисунке 11 изображена диаграмма последовательности взаимодействия пользователя и всех модулей приложения.

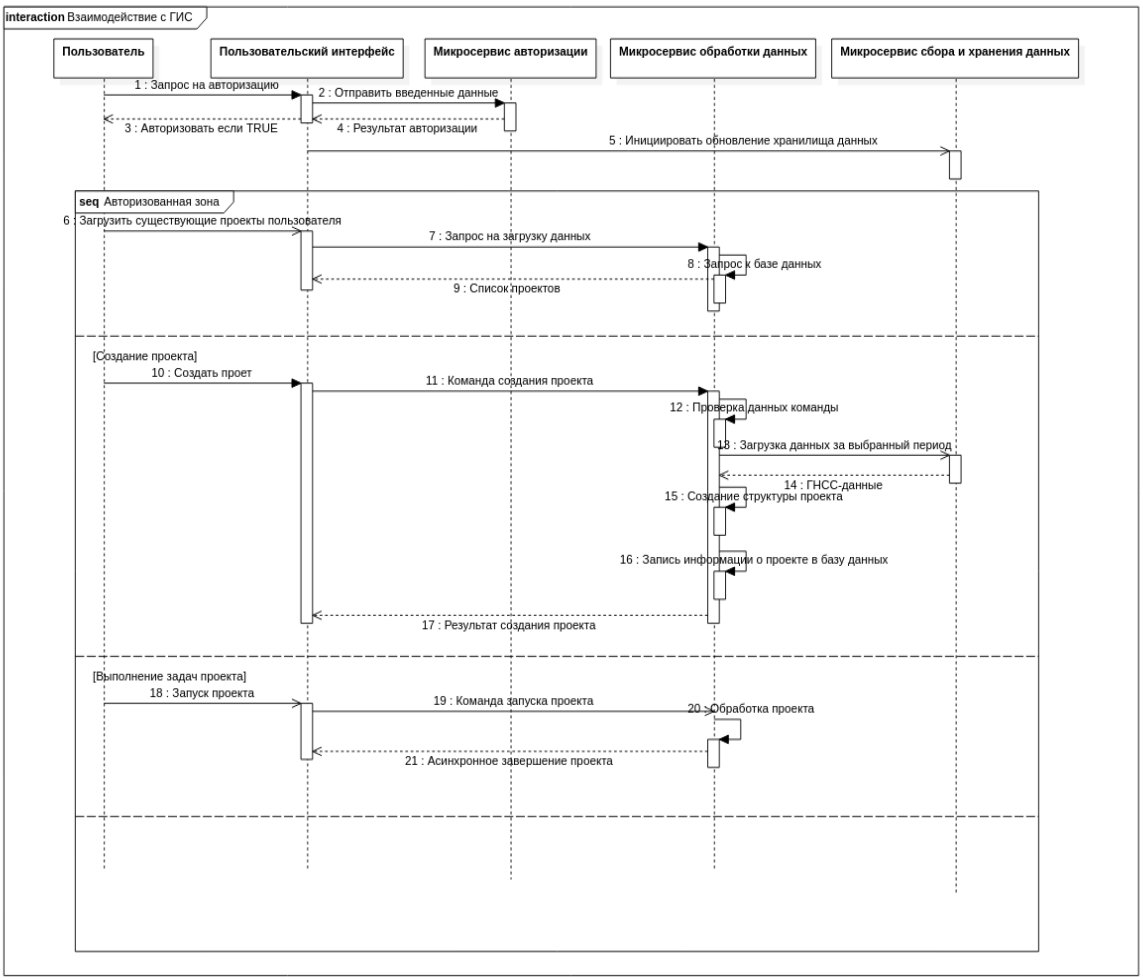

Рисунок 11 – Диаграмма последовательностей взаимодействия пользователя и приложения обработки ГНСС-данных

Так как архитектура приложения является микросервисной, то для описания внутренних взаимодействий, происходящих в ответ на пользовательские действия, необходимо изобразить все связанные микросервисы. Преимуществами диаграммы последовательностей, является то, что она очень наглядно отображает происходящие в приложении процессы. Временная линия направленна сверху вниз. Связи, возникающие в приложении, отображаются слева на право, за исключением тех случаев, когда микросервис выполняет внутренние операции в рамках внешнего запроса

2.4.2 Проектирование взаимодействия модулей в микросервисах

Для взаимодействия между внутренними частями микросервиса (программными модулями) используется паттерн Mediator, который сопоставляет

CQRS вызовы со сборками которые будут его обрабатывать, гарантирует выполнение всех звеньев цепочки обработки и возвращает ответ сборке-источнику CQRS вызова.

Микросервис систематизации, сбора и хранения ГНСС-данных

Микросервис сбора (функциональная модель представлена на рисунке 12) представлен в виде ASP NET Core приложения и выполняет функции систематизации и хранения информации, занимаются загрузкой, из различных источников, данных, необходимых для проведения расчетов, а также их первичной обработкой (конвертирование в открытый формат Rinex).

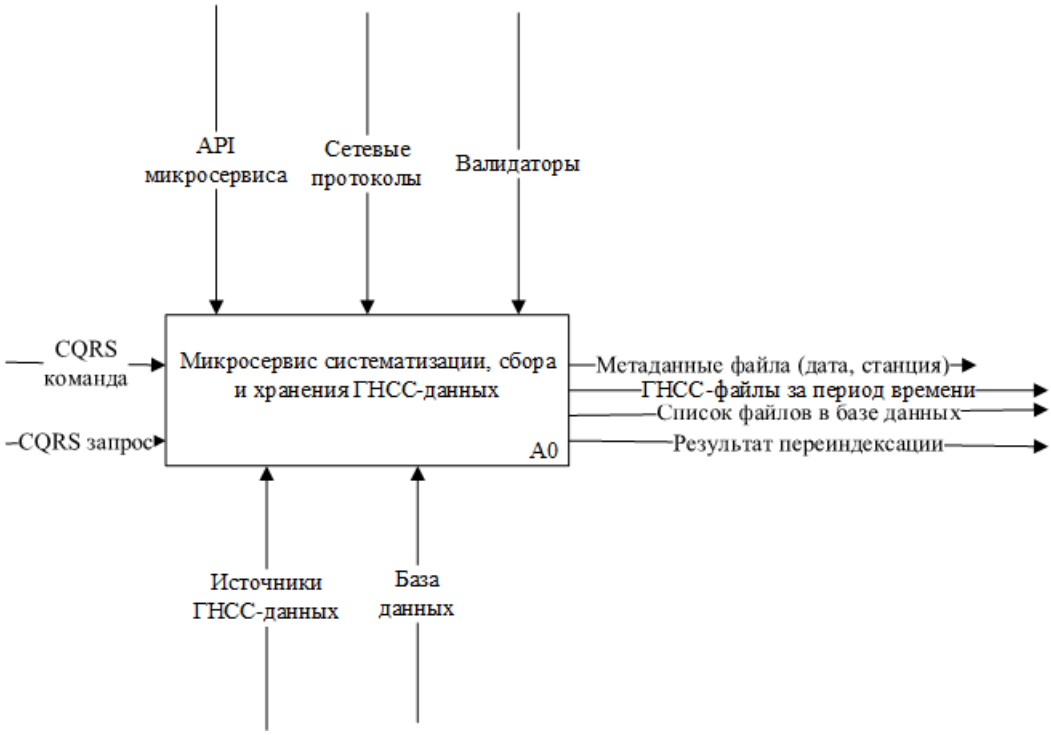

Рисунок 12 – Функциональная модель микросервиса систематизации, сбора и хранения в нотации IDEF0

На вход микросервис принимает CQRS команды и запросы. Примером CQRS команды является запрос на индексацию хранимых данных, а CQRS запроса – запрос файлов, соответствующих определенному периоду времени.

Для хранения данных, получаемых от ГНСС-станций, микросервис использует Mongo DB. MongoDB является не реляционной документориентированной СУБД. Её применение не вносит ограничений на типы данных, имеет

хорошую горизонтальную масштабируемость. Декомпозиция первого уровня отражена на рисунке 13.

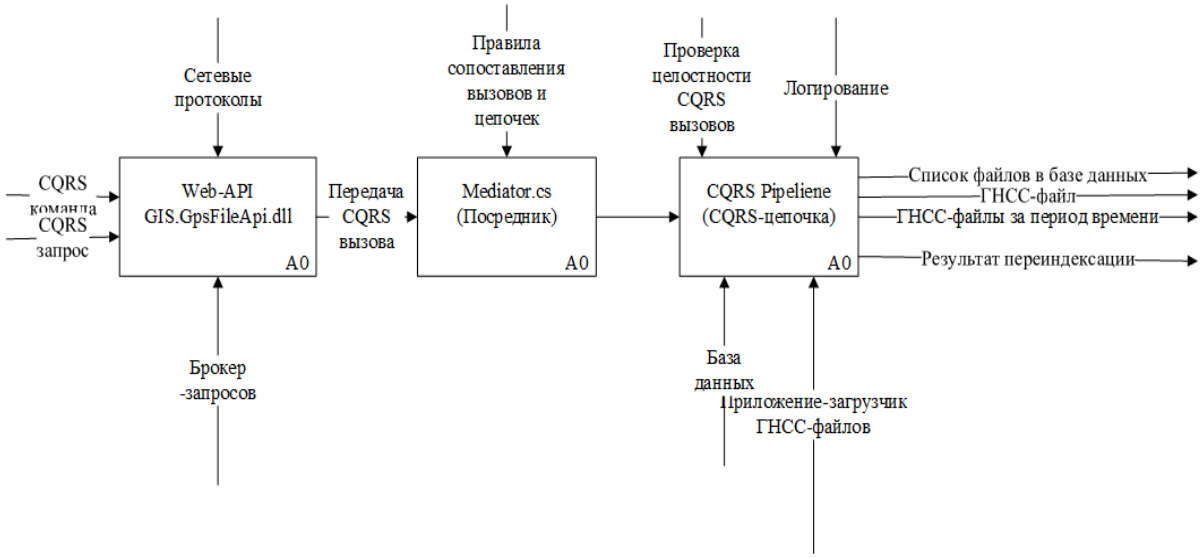

Рисунок 13 – Декомпозиция первого уровня функциональной модели микросервиса сбора и хранения ГНСС-данных в нотации IDEF0

Микросервис обработки данных

Функциональная модель микросервиса обработки представлена на рисунке 14. На вход микросервис принимает CQRS-запрос или команду (рисунок 14), которые на прикладном уровне представлены HTTP запросами: get, post, put.

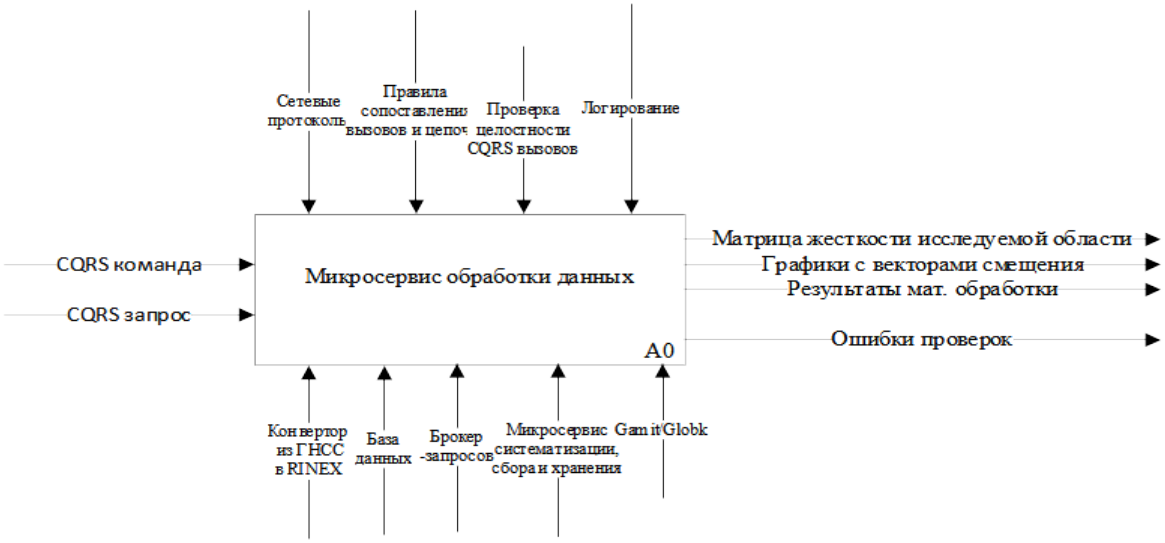

Рисунок 14 – Функциональная модель микросервиса обработки данных в нотации IDEF0

Обеспечивающими подсистемами микросервиса являются: конвертор из ГНСС в RINEX, база данных, брокер-запросов, микросервис сбора, систематизации и хранения данных, математический пакет GamitGlobk.

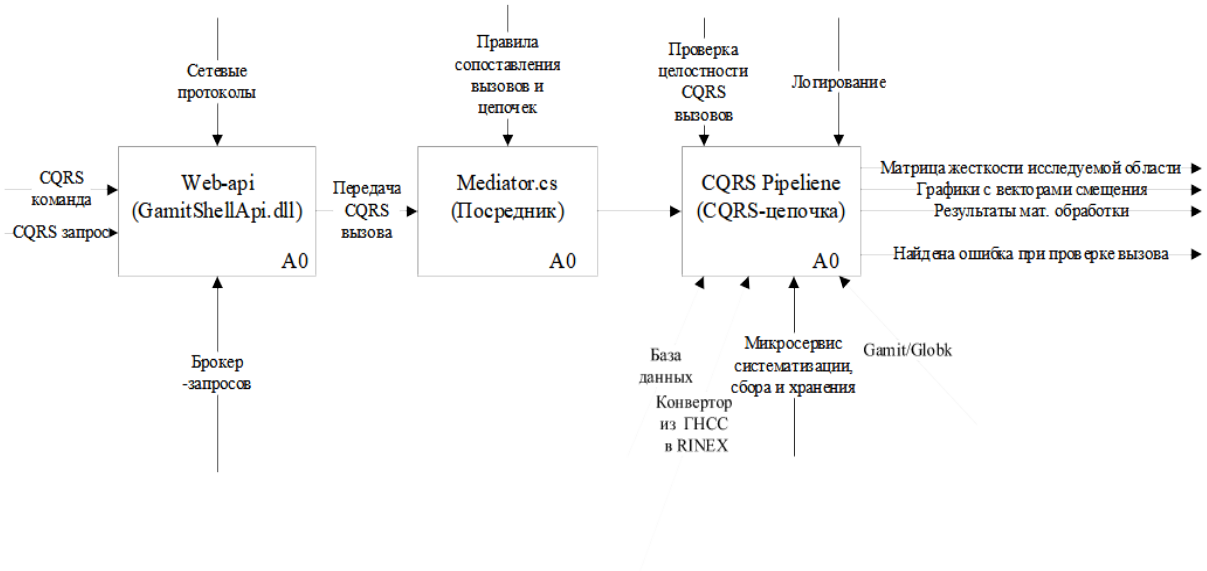

Рисунок 15 – Декомпозиция первого уровня функциональной модели микросервиса обработки данных в нотации IDEF0

Регуляторами работы модуля являются: сетевой-протокол (регламентирует взаимодействие по HTTP-протоколу), правила сопоставления вызовов и цепочек (маршрутизация команд, запросов и их обработчиков), валидатор целостности CQRS вызовов, логгирование. Декомпозируем модель, для более полного понимания внутреннего строения микросервиса. (рисунок 15).

В результате этапа проектирования была получена спецификация требований к программному обеспечению микросервиса обработки ГНСС-данных, которая определила границы проекта, ограничения дизайна и реализации, требования к продукту и основные функции.

Разрабатываемый «микросервис» является компонентом приложения обработки ГНСС-данных с применением геоинформационных технологий.

2.5 Алгоритм работы с входными данными

Каждый микросервис получает на вход HTTP-запросы, и преобразует их в соответствующие CQRS-запросы. Они передаются внутрь приложения, находя-

щегося в микросервисе и обрабатываются внутренними модулями с помощью «цепочек обработки». Для своей работы «микросервис» требует сетевые протоколы, правила сопоставления вызовов и цепочек, правила проверки целостности CQRS вызовов и модуль логирования. Обобщенный пример цепочки вызовов показан рисунке 16

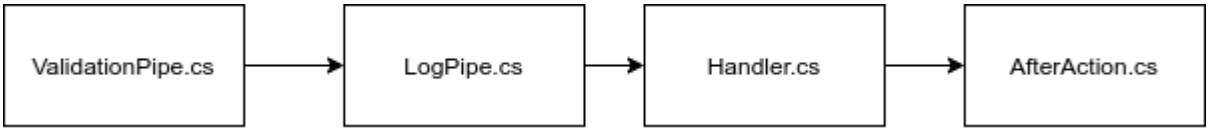

Рисунок 16 – Обобщенная «цепочка обработки» CQRS-вызова

Цепочка состоит из четырех элементов:

ValidationPipe.cs содержит логику проверки входящих вызовов (например: существование записи в базе данных, возможность выполнить операцию), и случает отрицательного результата генерирует исключение.

LogPipe.cs выполняет логирование входящих данных, благодаря чему упрощается процесс отладки работающего сервиса.

Handler.cs основной обработчик вызова, в зависимости от вызова выполняет определенную логику

AfterAction.cs выполняется после успешного выполнения вызова, что позволяет, например, обеспечить логирование результата, отправку уведомлений о завершении на почту и т. п.

2.6 Обоснование выбора средств реализации приложения

Исходя из требований к программному продукту и перечня функций, требуемых к реализации, средства разработки должны предоставлять возможность разработки как клиентской, так и серверной логики, иметь хорошую интеграцию с существующими реляционными базами данных, а также поддержку кроссплатформенных приложений. Так же, в соответствии с выбранной методологией разработки, инструментарий должен в полной мере обеспечивать поддержку разработки через тестирование.

Подходящими кандидатами являются следующие языки программирова-

ния:

- Python;
- Java;
- C#.

Для выбора наилучшего варианта сгруппируем особенности данных языков, связанные с решаемой задачи в таблицу. Важными особенностями в разработке модулей для микросервисов являются: надежность, универсальность, удобство запросов к базе данных, кроссплатформенность и быстродействие.

Таблица 5 – Сравнение инструментов разработки

Наименование	Вариант 1	Вариант 2	Вариант 3
Язык программирования	C#	Python	Java
Кроссплатформенность	да	не полная	да
Поддержка Unit тестов	да	да	да
Взаимодействие с БД	Entity Framework Core 6	SQL Alchemy	Hibirnate
Клиент-серверное взаимодействие	да	да	да
Среда	.net core 3.1/5.0	Python 3.7	Java 11
СУБД	любая	любая	любая
IDE	VS 2019, VS Code, JetBrains Rider	PyCharm, VS Code	IntelliJ IDEA

Требование кроссплатформенности исходит из необходимости исключить зависимость компонента серверной логики и базы данных от конкретного семейства операционных систем (Windows, Unix), так как такая зависимость будет дорогостоящей при дальнейшем развитии и поддержке программного обеспечения. Сравнение средств разработки приведено в таблице N.

Среда .NET Core 3.1 предоставляет доступ к таким мощным языкам как C#, F#, C++/CLI, VB .Net. [7] и является полностью кроссплатформенной, а использование стандарта .NET Standart 2.1 позволяет создавать кроссплатформенные библиотеки совместимые с различными средами .NET.

Java является известным и мощным языком программирования, для работы с базой данных существует ORM Hibirnate, однако, значимыми факторами, являются: отсутствие у команды навыков в этой экосистеме и высокая стои-

мость IDE для комфортной и эффективной работы с Java, стоимость IntelliJ IDEA 3350 руб./месяц или же 33500 руб. в год за одну лицензию.

Python современный кроссплатформенный язык. В его состав по умолчанию не входит какая-либо ORM система для работы с БД, однако существуют сторонние расширения такие как SQL Alchemy. К недостаткам таких решений можно отнести их узкую направленность (SQL Alchemy работает только с MySQL) и открытую кодовую базу, которая может кардинально изменяться с течением времени, что может затруднить поддержку существующего кода. Так же, полная кроссплатформенность не является частью языка по умолчанию, и, к примеру, разработка клиентской логики под операционную систему Android может стать довольно затруднительной. Однако, главным и существенным недостатком является низкая производительность кода написанного на Python. Существующие методы оптимизации такого кода, не являются универсальными и простыми, часто требуют довольно глубоких знаний в некоторых библиотеках из мира C++ (например OpenMP). Данный фактор значительно осложнит поддержку кода в долгосрочной перспективе.

Таким образом, наиболее подходящим решением является разработка программы на базе среды .NET Core с использованием ЯП C#.

2.6.1 Механизм взаимодействия с СУБД

ORM взаимодействие

ORM (Object Relational Mapping) – объектно реляционное отображение, технология, которая позволяет связать реляционную базу с концепциями объектно-ориентированных языков.

В рамках выбранной платформа, для работы с БД используется ORM Entity Framework Core 3.1 (версия EFC3.1 для платформы .NET Core). Это позволяет взаимодействовать с таблицами базы данных как с обычными моделями (классами). Так же для расширения функционала используется LINQ (Language Integrated Query) – язык интегрированных запросов, позволяющий одинаково взаимодействовать с любыми перечисляемыми объектами (LINQ to List, Array, Entities).

Так же EF Core 3.1 поддерживает любые реляционные СУБД. Т. е. код приложения, написанный с помощью EF и LINQ, не зависит от конкретной СУБД, что позволяет менять конкретную СУБД по требованию заказчика.

SQL взаимодействие

Взаимодействие с СУБД можно построить на основе классического языка запросов SQL. К достоинствам данного метода можно отнести его быстродействие и возможность писать оптимизированные запросы. Недостатки: привязка к определенной СУБД (разные СУБД используют различные диалекты языка SQL), временные затраты, отсутствие простого пути для отражения таблиц на объектно ориентированные сущности.

2.6.2 Механизм реализации требования модульности

MediatR

MediatR – библиотека, реализующая паттерн Mediator (посредник). Посредник – это поведенческий паттерн проектирования, который позволяет уменьшить связанность множества классов (модулей) между собой.

Данная библиотека применяется, для соблюдения требования модульности в рамках каждого отдельного микросервиса. Посредник позволяет разделить приложение на несколько сборок, например, сборка App.Core будет содержать только базовые вещи, используемые во всех модулях приложения. К таким «вещам» относятся: интерфейсы и абстрактные классы, классы-описания доменной области, сущности базы данных. Сборка App.BaseLogic будет содержать базовую бизнес логику для всех модулей. А сборка App.Validators будет содержать всевозможные сервисы-валидаторы. Для объединения сборок с помощью медиатора, на верхнем уровне приложения необходимо добавить все используемые сборки в анализируемые для медиатора, после чего он сможет находить и связывать компоненты в них.

Для реализации данного подхода необходимо реализовать несколько обязательных вещей, а именно: запрос и обработчик запроса. Так же, для реализации доступна цепочка и уведомление. Запрос представляет из себя некий poco объект, содержащий в себе всю необходимую для обработчика запроса инфор-

мацию, поступающую извне. Цепочка, позволяет создавать некую последовательность действий, которую необходимо выполнить после получения запроса, но до его обработки.

Классическая модульная структура

Классическая модульная структура предполагает разбиение программы на структурные блоки (модули), каждый из которых реализует конечное количество единых по смыслу частей. К преимуществам можно отнести отсутствие накладных расходов на работу посредника. К недостаткам – отсутствие единой цельной структуры. MediatR вводит определенное количество определений в рамках который программисты должны действовать (в программной среде выражается необходимостью реализовывать интерфейсы для работы MediatR), данное свойство позволяет сохранять четкую структуру на протяжении всего цикла разработки.

2.6.3 Выбор веб-фреймворка

Основой веб-приложения, создаваемого на платформе .NET Core является веб-фреймворк ASP NET Core. Данный фреймворк разрабатывается и поддерживается сообществом и компанией Microsoft. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кроссплатформенной среды .NET Core, которая может быть развернута на основных популярных операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core мы можем создавать кроссплатформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кроссплатформенный веб-сервер Kestrel.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

- новый легковесный и модульный конвейер HTTP-запросов;

- возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса;
- использование платформы .NET Core и ее функциональности;
- распространение пакетов платформы через NuGet;
- интегрированная поддержка для создания и использования пакетов NuGet;
- единый стек веб-разработки, сочетающий Web UI и Web API;
- конфигурация для упрощенного использования в облаке;
- встроенная поддержка для внедрения зависимостей;
- расширяемость
- кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux;
- развитие как open source, открытость к изменениям.

2.6.4 Механизм контейнеризации

Микросервисная архитектура, в отличие от монолитной, предполагает создание и запуск нескольких отдельных приложений, независимых друг от друга. Каждое из этих приложений реализует свою часть функций системы. Так же, каждое из этих приложений может быть расположено на разных физических машинах и даже на серверах, расположенных в разных уголках земли. Для того чтобы описать программные требования и зависимости конкретного приложения, а также автоматизировать процессы сборки и публикации таких приложений используются технологии контейнеризации, например LXC.

LXC/LXD

LXC/LXD (Linux Container) – это система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров операционной системы Linux на одном физическом узле. LXC не использует виртуальные машины, а создаёт виртуальное окружение с собственным пространством процессов и сетевым стеком. Все экземпляры LXC используют один экземпляр ядра операционной системы. Данная технология применяется системой управления контейнерами Docker.

Docker

Docker – это система управления контейнерами, в своей основе использует файлы Dockerfile для описания процессов упаковки приложения. Данная технология позволяет описать процессы сборки приложения (удобно для описания специфичных зависимостей приложения), а также сформировать финальный образ приложения, который можно запустить простой командой в любой среде.

Так, например, если итоговое приложение состоит из нескольких различных докер-образов, необходим инструмент, который позволит управлять их совместным развертыванием, запуском, установкой. Таким инструментом является docker-compose. Docker-compose это инструмент, используемый для определения и запуска нескольких приложений контейнеров, файл docker-compose.yml используется для определения необходимых для приложения сервисов (например приложению необходима СУБД MySQL).

2.6.5 Математический пакет

Для обработки ГНСС-данных существует несколько математических пакетов. Каждый из них имеет свои достоинства и недостатки.

Математический пакет GAMIT/GLOBK

Данный пакет, разработанный в Массачусетском технологическом университете, представляет из себя консольную утилиту, состоящую из множества маленьких прикладных программ [15] (рисунок 17).

Каждая утилита реализует определенный функционал (расчет поправок для спутниковых орбит, фильтрация данных в ГНСС-файлах (удаление выбросов в выборке данных) и другие).

Так же пакет содержит информацию о производителях станций и антеная. Это позволяет использовать дополнительные корректировки при расчетах.

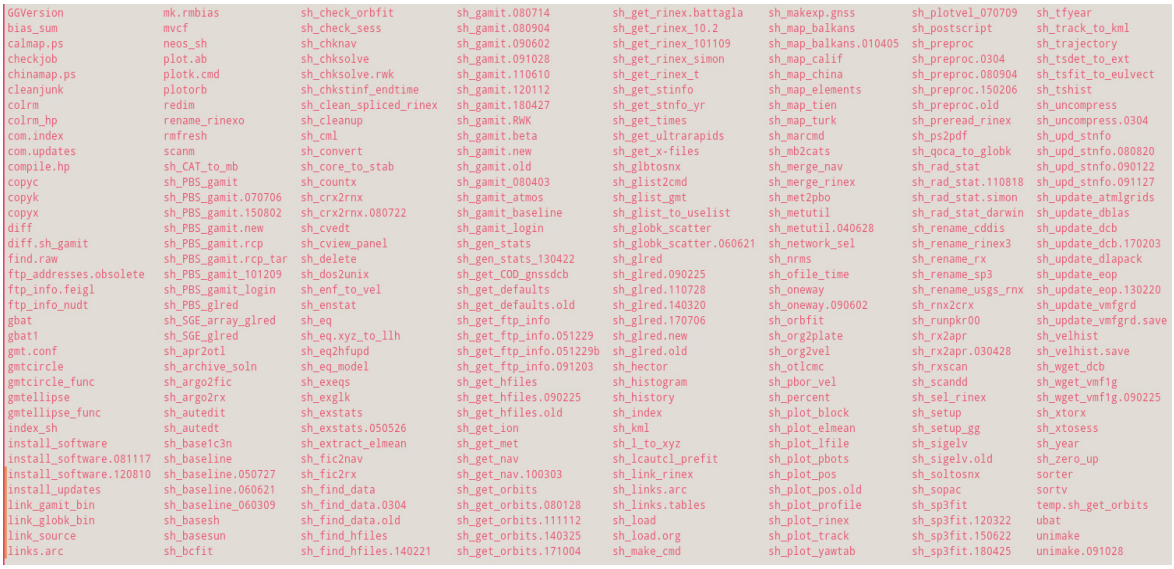

Рисунок 17 – Математический пакет GAMIT/GLOBK

К преимуществам данного пакета можно отнести: бесплатность, наличие интерфейса командной строки, известность и подтвержденная годами расчетов, адекватность получаемых результатов.

Математический пакет Bernese GNSS Software

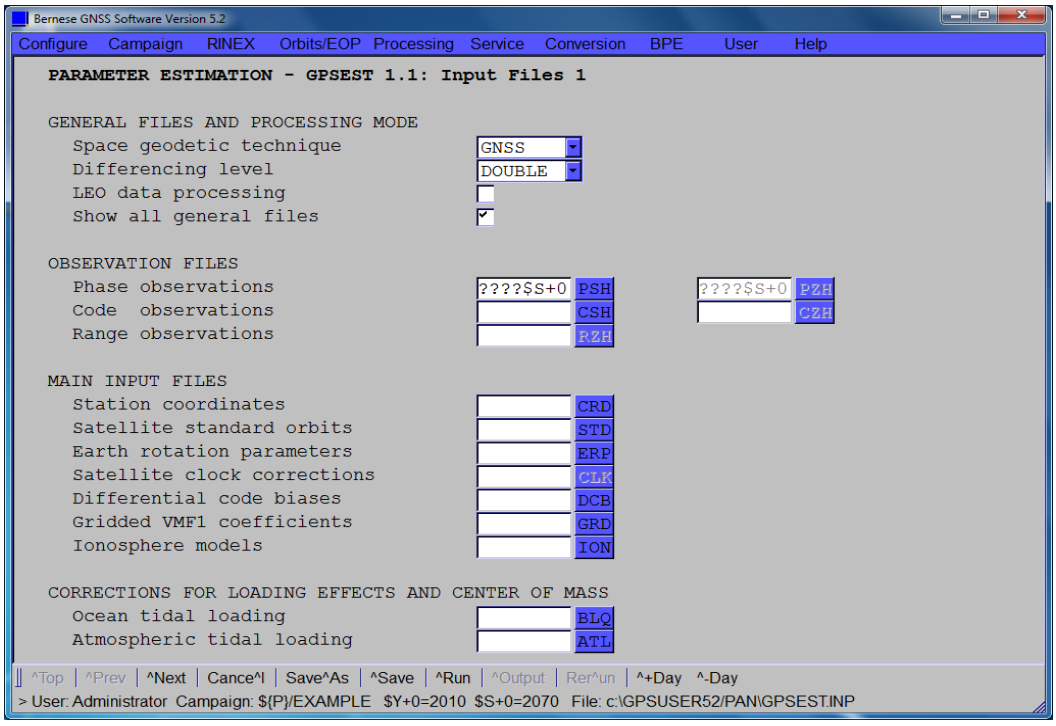

Рисунок 18 – Интерфейс Bernese GNSS Software

Данное программное обеспечение разработано в Швейцарии, в Институте

Астрономии Университета Берн (AIUB). В отличае от GAMIT/GLOBK является платным, но в свою очередь предоставляет хоть и устаревший, но графический интерфейс пользователя (рисунок 18). Основными же проблемами данного программного продукта являются: его стоимость в 12000 CHF, что примерно равняется 1 миллиону российских рублей, закрытость исходного кода и отсутствие внешнего интерфейса, что не позволяет автоматизировать работу специалистов.

2.7 Характеристика выбранного программно-технического обеспечения

Характеристика среды разработки

Разработка приложения ведется с использованием операционной системы семейства Linux и IDE от компании JetBrains – Rider (рисунок 19).Rider – это кроссплатформенная интегрированная среда разработки программного обеспечения для платформы .NET, разрабатываемая компанией JetBrains. Поддерживаются языки программирования C#, VB.NET и F#.

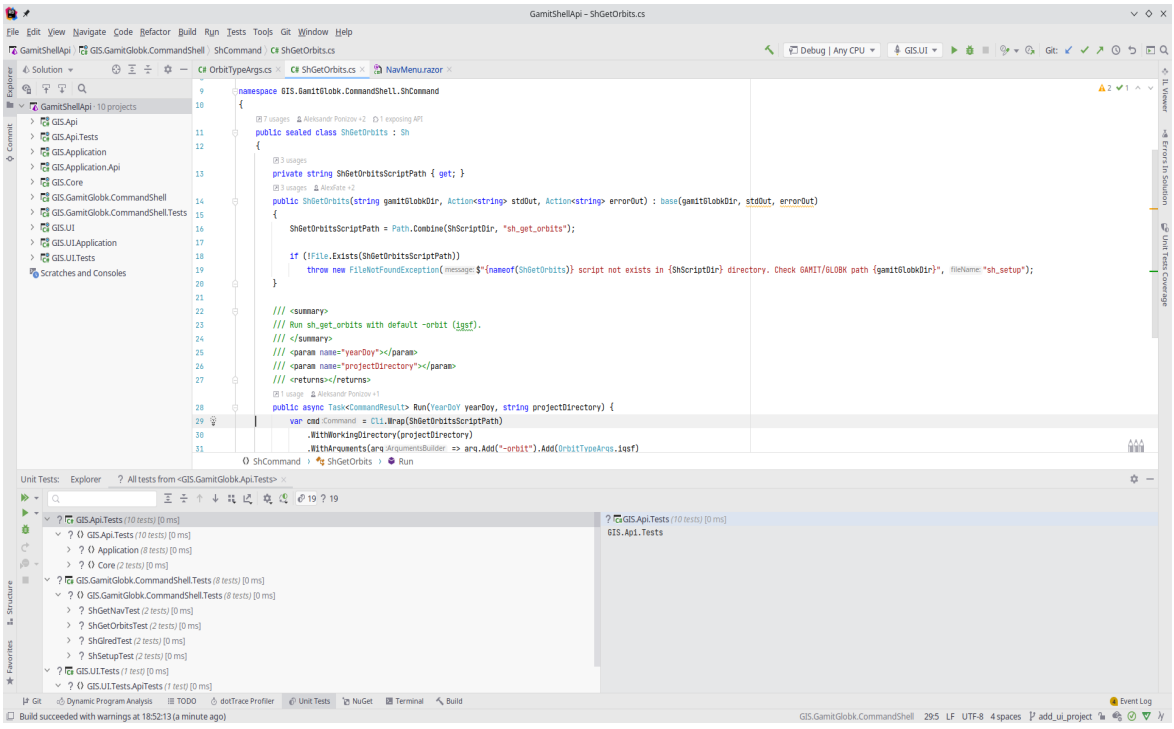

Рисунок 19 – Среда разработки JetBrains Rider

Данная среда удовлетворяет все потребности разработчика при разработке современных приложений, имеет возможность запуска модульных тестов, от-

ладки приложения внутри docker-контейнера, анализ покрытия приложения тестами (рисунок 20), профилирования проблем производительности.

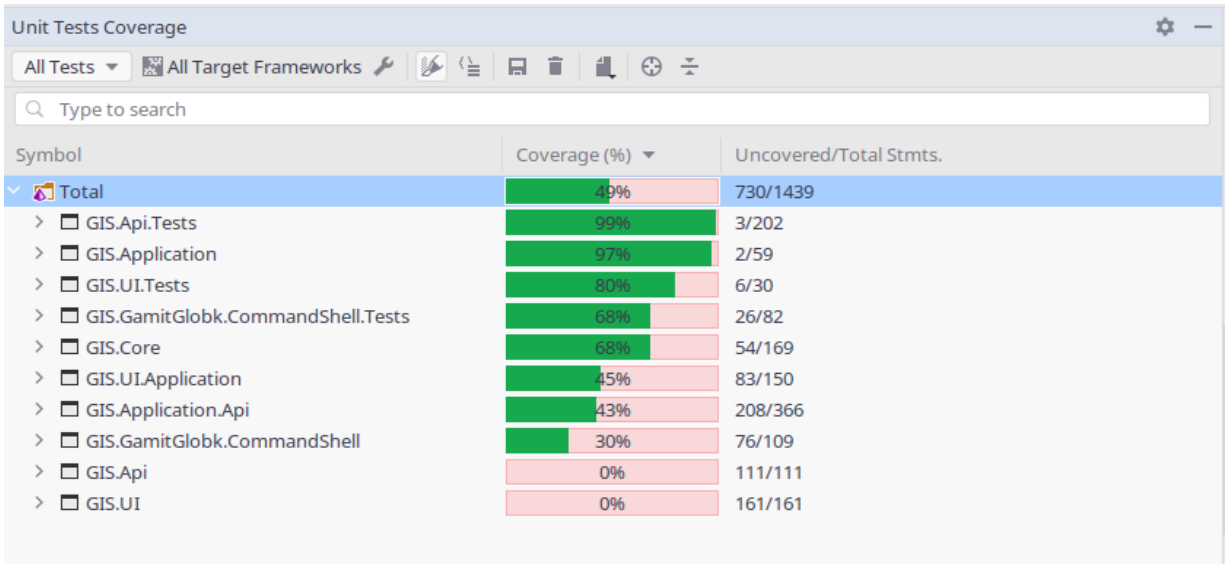

Рисунок 20 – Анализ площади покрытия тестами

Площадь покрытия тестами это важный показатель, который позволяет разработчику оценить объем кода непокрытого тестами. Код непокрытый тестами это потенциальный источник проблем и ошибок.

Характеристика аппаратного обеспечения

Для запуска разрабатываемого программного обеспечения необходим x64 процессор: с поддержкой визуализации для операционных систем семейства Windows/MacOS; без поддержки в случае использования операционной системы семейства Linux.

Оперативной памяти не менее 4 ГБ.

Места на жестком диске: 20 ГБ и более.

Сетевое соединение с каналом 1 Гбит/с при распределенном размещении микросервисов.

Программно-техническое обеспечение

Для разработки микросервисов, реализующих основные алгоритмы приложения, ранее было решено использовать технологию «контейнеризации», примерами которой являются технологии docker и lc (linux container). Техноло-

гия «контейнеризации» является кроссплатформенной (docker поддерживает Windows и MacOS), однако, её нативная поддержка реализована только в операционных системах семейства Linux.

Однако, использование представителей серверных операционных систем: Ubuntu Server 18.04 или Debian 10, в виде как есть, показало себя не с лучшей стороны. Основной недостаток сводился к большим временным затратам на администрирование групп виртуальных машин и контейнеров, расположенных на различных физических серверах. Поэтому решено было использовать Proxmox Virtual Environment – дистрибутив, базирующийся на стабильной ветке Debian, предлагающий пользователям: свежее ядро Linux (для поддержки новых процессоров и оборудования), графическое веб-окружение для администрирования и управления, поддержку кластеризации, резервное копирование виртуальных машин и контейнеров.

Первым этапом являлась установка и настройка Proxmox VE на сервера. Сервера в количестве трех штук имеют следующие аппаратные характеристики:

Процессор: 2 x Intel Xeon 24 core

ОЗУ: 64 Гб

ROM: от 10 ТБ

Существует два варианта установки ProxmoxVE:

- обновление дистрибутива Debian 10 до Proxmox VE. Выполняется посредством добавления репозиториев proxmox в Debian и отключение стандартных репозиторив Debian;
- чистая установка дистрибутива PVE.

Следующим этапом является объединение физических серверов (нод) в датацентр (кластер). Это необходимо, для централизации управления серверами, быстрого переноса виртуальных машин и контейнеров между нодами. В результате получаем настроенный датацентр.

Функции Proxmox VE

Proxmox VE предоставляет возможность настраивать брандмауэр отдельно для каждого контейнера и виртуальной машины (рисунок 21).

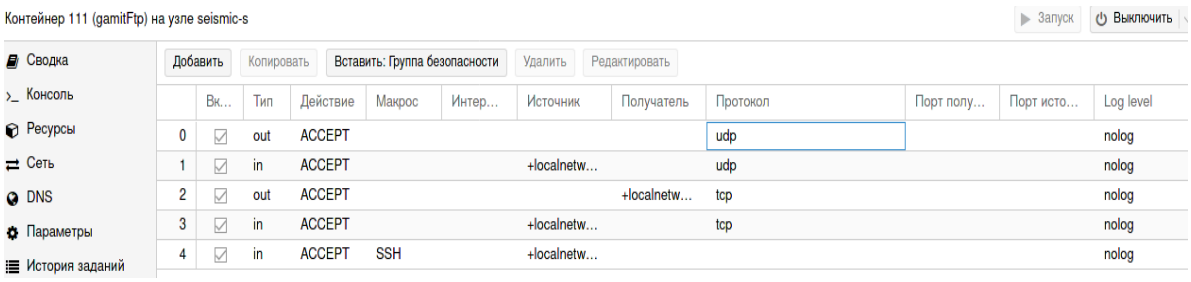

Рисунок 21 – Настройки брандмауэра для контейнера gamitApi и stationApi

На рисунке 21 видим настроенные разрешающие правила брандмауэра, по умолчанию брандмауэр разрешает исходящий трафик и запрещает входящий. Настройка брандмауэра позволяет обеспечить безопасность передачи данных в локальной сети, а так же ограничить доступ к контейнерам. Например, для обмена данными между браузером пользователя и микросервисом графического интерфейса необходимо открыть 443 порт (HTTP over ssl). А для взаимодействия микросервиса с брокером запросов должен быть разрешен трафик по 5672 TCP/UDP порту.

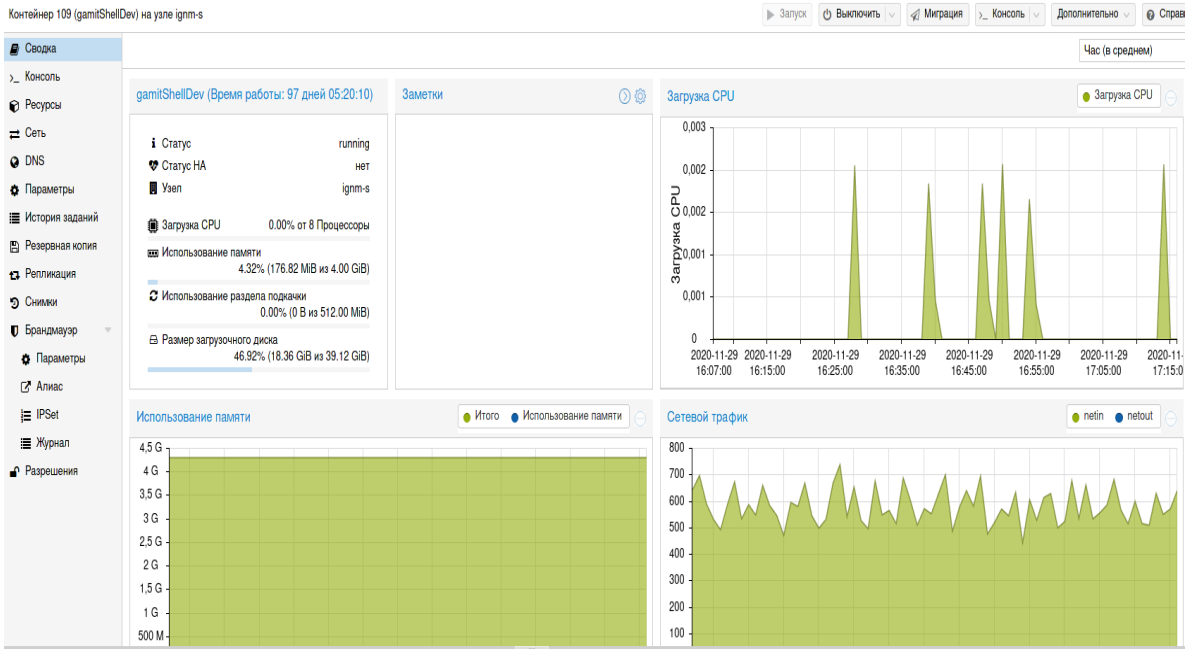

Рисунок 22 – Статистика работы микросервиса обработки данных

На рисунке 22, видно, что Proxmox VE позволяет просматривать статистику по каждому элементу системы. Возможно просмотреть потребление ресур-

сов за неделю, месяц, год. Возможен показ средних или максимальных данных по каждому ресурсу (IO, загрузка cpu, использование ОЗУ, сетевой трафик). Так же возможно логгирование входящего и исходящего сетевого трафика.

3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ ОБРАБОТКИ ГНСС-ДАННЫХ

3.1 Основные этапы практической разработки программного продукта

3.1.1 Выбор методологии разработки приложения

Перед началом этапа проектирования необходимо определить методологию разработки программного обеспечения. Методология разработки зависит от объема планируемых работ. Оптимальными являются методологии, которые смогут обеспечить качество и скорость разработки программного продукта, а также гибкость к изменениям требований.

Одной из таких методологий является «Экстремальное программирование» (XP) (рисунок 23) [16]. Она включает в себя следующие особенности, которые позволят полноценно реализовать требования к продукту:

Команда – единое целое. Все участники проекта, где применяется практика экстремального программирования работают как одна команда. В такую команду обязательно должны входить: представитель заказчика (в идеале реальных конечный пользователь программного продукта); бизнес-аналитики, разработчики, тестировщики, коуч и менеджер проекта, который обеспечивает команду ресурсами.

Двухэтапное планирование. Первым этапом является планирование релиза. При планировании релиза команда программистов встречается с заказчиком для выяснения какая функциональность должна быть реализована к следующему релизу. После этого заказчик определяет приоритет для каждой из задач. Разработчики, в свою очередь, дают временную оценку каждой задаче. Заказчик просматривает эти карточки и дает добро на начало работы. В случае, если команда не успевает реализовать все требования к релизу, релиз не переносится, а режится часть функционала менее приоритетная для заказчика.

Частые релизы. Версии в данной методологии выпускаются часто, но с маленьким объемом новых функций и исправлений. Во-первых, это облегчает

процессы тестирования. Во-вторых, каждую итерацию заказчик получает часть функционала, несущую бизнес-логику.

Пользовательские тесты. Заказчик определяет приемочные тесты, которые проверяют работоспособность очередной функции продукта. Команда пишет эти тесты и использует их для тестирование готового кода.

Короткий цикл обратной связи включает в себя разработку через тестирование и парное программирование, эти компоненты способствуют увеличению качества выходного продукта.

Рисунок 23 – Практики «Экстремального программирования»

Непрерывный процесс включает в себя непрерывную интеграцию, рефакторинг и небольшие релизы. Непрерывная интеграция и небольшие релизы позволят увидеть реальную полезность и подтвердить необходимость разработки продукта еще на ранних этапах. А рефакторинг кода обеспечит качественную кодовую базу, которую будет легко поддерживать и модифицировать.

Стандарты кодирования. Когда над кодом работает множество программистов должны быть приняты единые стандарты оформления кода.

Разработка, основанная на тестировании. Тесты пишутся программистами, причем они доджны быть написаны до написания кода, который нужно про-

тестировать. При таком подходе получается код, покрытый тестами на значение близкое к 100 %.

Простой дизайн. Простой дизайн предполагает использование принципа KISS (keep it simple stupid – «Делай проще, тупица»). Данный принцип утверждает, что большинство систем работает лучше, если они остаются простыми и не усложняются.

Рефакторинг. Код должен постоянно подвергаться рефакторингу. Рефакторинг – это процесс улучшения дизайна системы, с целью приведения его в соответствие новым требованиям. Рефакторинг включает в себя удаление дублей кода, повышение связанности и снижения сопряжения.

В качестве модели жизненного цикла используется спиральная модель, которая сочетает идеи итеративной и каскадной моделей. Суть ее в том, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

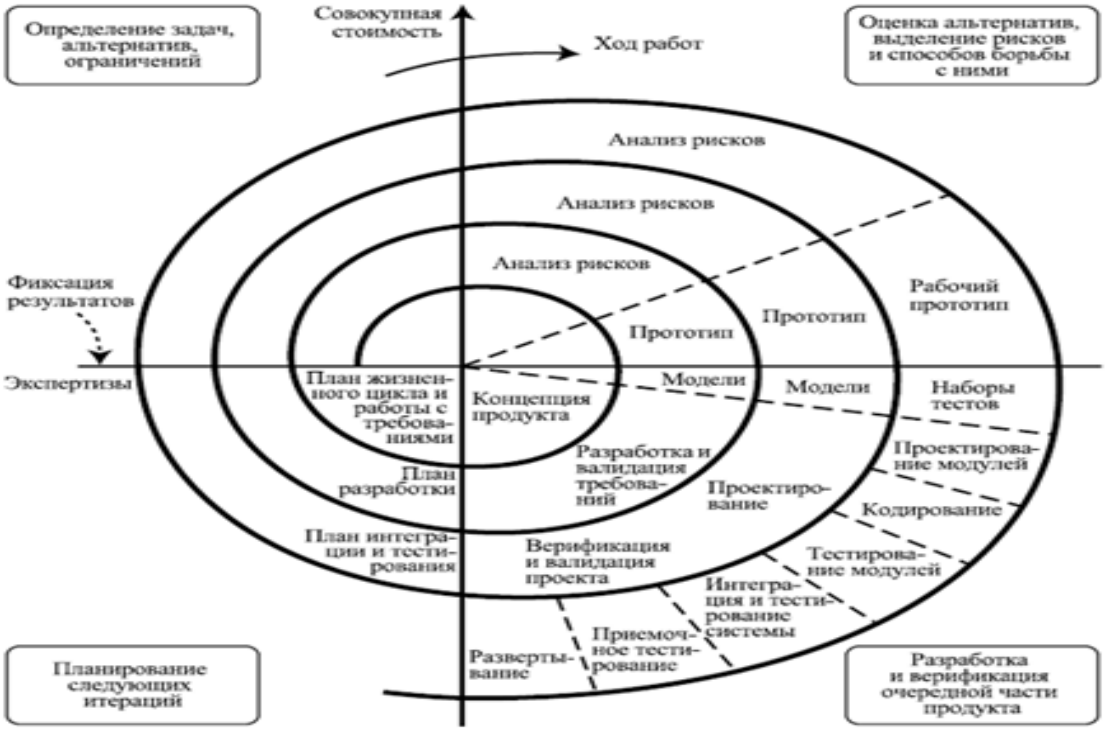

Рисунок 24 – Визуальное представление спирального

В спиральной модели жизненный путь разрабатываемого продукта изображается в виде спирали, которая, начавшись на этапе планирования, раскручивается с прохождением каждого следующего шага. Таким образом, на выходе из очередного витка мы должны получить готовый протестированный прототип, который дополняет имеющийся результат. Прототип, удовлетворяющий всем требованиям – готов к релизу.

Главная особенность спиральной модели – концентрация на возможных рисках. Для их оценки даже выделена соответствующая стадия.

Основные типы рисков, которые могут возникнуть в процессе разработки ПО:

- нереалистичный бюджет и сроки;
- дефицит специалистов;
- частые изменения требований;
- чрезмерная оптимизация;
- низкая производительность системы;
- несоответствие уровня квалификации специалистов разных отделов.

Плюсы спиральной модели:

- улучшенный анализ рисков;
- хорошая документация процесса разработки;
- гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;
- раннее создание рабочих прототипов.

Недостатки спиральной модели:

- может быть достаточно дорогой в использовании;
- управление рисками требует привлечения высококлассных специалистов;
- успех процесса в большой степени зависит от стадии анализа рисков;
- не подходит для небольших проектов.

Сочетание спиральной модели жизненного цикла и подходов «экстремального программирования» создают для проекта условия, в которых изменение

требований не является чем, то проблемным. На каждой итерации жизненного цикла происходит покрытие нового кода тестами и обновление проектной документации в соответствии с текущими реалиями.

3.1.2 Разработка модуля сбора ГНСС-данных

Функциональная модель приложения показана на рисунке 25. Оно не поставляется отдельным микросервисом, но применяется в микросервисе сбора и хранения данных.

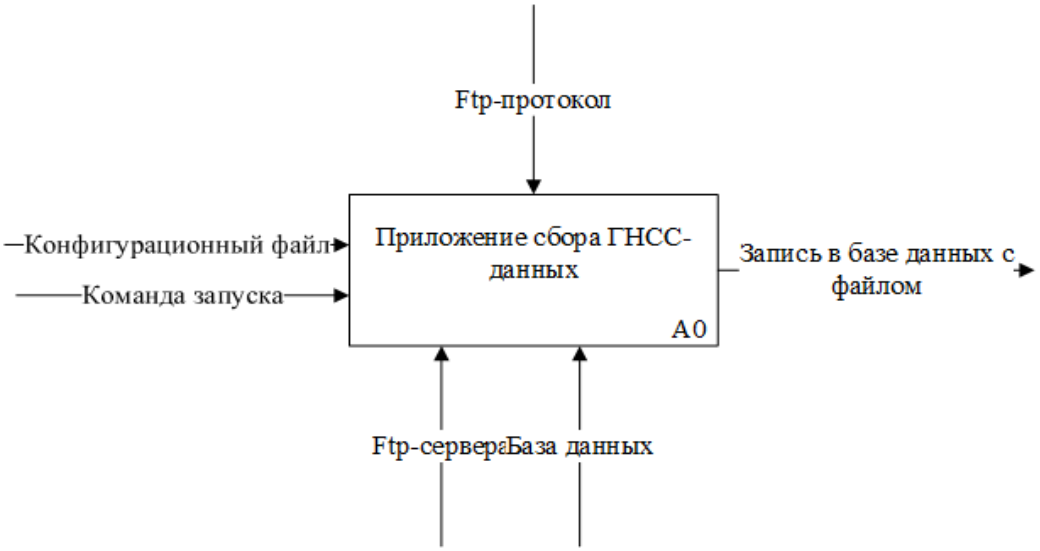

Рисунок 25 – Функциональная модель приложения сбора ГНСС-данных.

Основной задачей данного приложения является загрузка ГНСС-данных с ftp серверов, указанных в конфигурационном файле. Так же приложению необходимо указать шаблон путей и имен файлов, которые необходимо загрузить.

3.1.3 Разработка модуля конвертирования ГНСС-файлов в RINEX формат

Приложение конвертирования не поставляется отдельным микросервисом, но применяется в микросервисе обработки данных. Его основной задачей является преобразование проприетарных, бинарных ГНСС-файлов, получаемых со станций в Rinex-файлы.

RINEX (сокр. англ. Receiver Independent Exchange Format) – формат обмена данными для файлов исходных данных спутниковых навигационных приём-

ников. Он позволяет пользователям производить пост-обработку полученных данных для выполнения более точных вычислений – обычно с помощью других данных, неизвестных приемнику, например за счёт применения более точной модели атмосферных параметров в момент измерений.

Выходные данные навигационного приёмника представляют собой его координаты, скорость, время и другие характеристики. Однако вычисление этих величин основаны на серии измерений, полученных от одного или более спутниковых созвездий. Хотя приёмники вычисляют свои координаты в режиме реального времени (во время измерения), во многих случаях промежуточные данные измерений полезно сохранять для дальнейшего использования.

RINEX – это стандартный формат, который позволяет хранить и передавать промежуточные измерения, произведенные приёмником, а также проводить постобработку полученных данных различными приложениями различных производителей приёмников и программ.

Формат RINEX спроектирован так, чтобы его можно было дополнять со временем, адаптировать под новые типы измеряемых данных и новые спутниковые навигационные системы. Наиболее распространенная в настоящее время версия 2.11, в которой содержатся данные о псевдодальности, фазе несущей и Доплеровском сдвиге частот для GPS или ГЛОНАСС, совместно с данными от систем спутниковой дифференциальной коррекции EGNOS и WAAS.

Последняя на сегодня версия формата, 3.04, поддерживает все публично доступные сигналы: GPS, ГЛОНАСС, Галилео, китайскую Бэйдоу, японскую QZSS и индийскую IRNSS системы навигации. Кроме того, RINEX 3.04 позволяет использовать обновление GLONASS CDMA, а также новые сигналы BeiDou III и QZSS II

Структурно, приложение конвертирования данных включает в себя следующие подмодули:

- ToDatConverter.cs – модуль конвертирования проприетарных ГНСС-файлов в промежуточные бинарные файлы формата .dat;
- ToRinexConverter.cs – модуль конвертирования промежуточных .dat

файлов в Rinex файлы;

- FileConverterProvider.cs – модуль агрегирующий функционал предыдущий двух модулей. Использует функционал библиотеки TPL Dataflow для распараллеливания обработки каждого файла;

- Helpers.dll – модуль реализует вспомогательные функции, такие как сортировка файлов по дням и станциям, удаление промежуточных файлов, обработка «разорванных сессий»;

- Core.dll – модуль поставляет основные типы, интерфейсы и абстрактные классы, используемые в других модулях.

Первая версия приложения реализовывала синхронный процесс обработки ГНСС-файлов (рисунок 26).

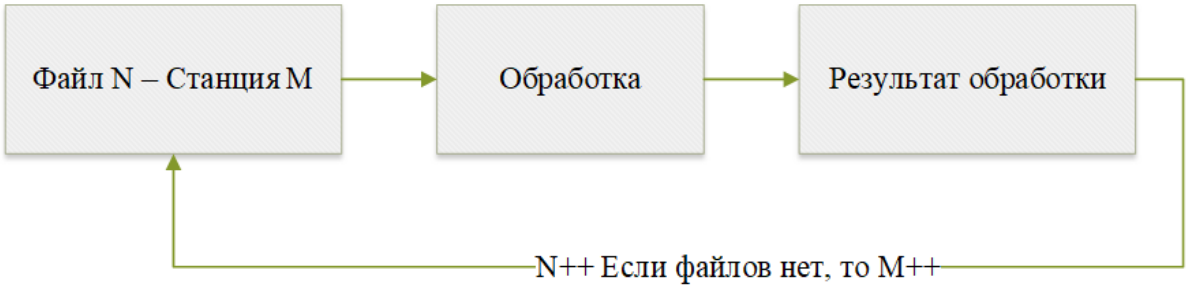

Рисунок 26 – Схема синхронной работы приложения обработки ГНСС-данных

Файлы, предназначенные для обработки, располагаются в древовидном структуре файловой системы. Корень включает в себя папки название которых соответствует названию станций. В папках расположены ГНСС-файлы. При запуске приложения выполняется сортировка файлов по дням года. После чего программа выполняет последовательный обход файлов в каждом дне для каждой станции.

К недостаткам синхронного процесса можно отнести большие временные затраты. Особенностью процесса конвертирования ГНСС-файлов является их большое увеличение в объеме. Файл размером в 20 Мб, становится RINEX-файлом размером 250 Мб.

Для уменьшения временных затрат на конвертирование было принято решение использовать механизмы распараллеливания, предложенные платформой

.NET Core. Таких механизма два: один, расположенный в пространстве имен System.Threading.Tasks класс Parallel; второй, расположенный в том же пространстве имен клас DataFlow.

TPL DataFlow (поток данных, библиотека параллельных задач). Библиотека параллельных задач (TPL) предоставляет компоненты потока данных, что позволяет повысить надежность приложений с включенным параллелизмом. Эти компоненты потока данных вместе называются *Библиотекой потоков данных TPL*. Эта модель потоков данных поддерживает программирование на основе субъектов путем обеспечения внутрипроцессной передачи сообщений для недетализированного потока данных и задач по конвейеризации. Компоненты потоков данных строятся на типах и инфраструктуре планирования TPL и интегрированы с языковой поддержкой асинхронного программирования на C#, Visual Basic и F#. Эти компоненты потоков данных полезны при наличии нескольких операций, которые должны асинхронно взаимодействовать друг с другом, или при необходимости обрабатывать данные по мере того, как они становятся доступными.

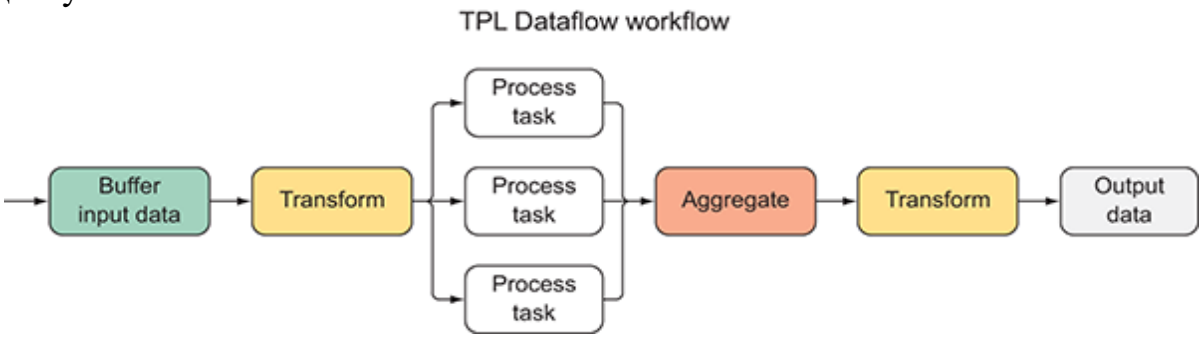

Рисунок 27 – Параллелизм, предлагаемый библиотекой параллельных задач

Параллелизм TLP Dataflow предполагает, что есть некий асинхронный поток входных данных, который мы можем сохранять в буфере. Для этих данных выстраивается «поток данных», который представляет из себя набор независимых обработчиков и узлов агрегации (рисунок 27).

TPL Dataflow реализует параллелизм посредством асинхронности, где во время ожидания чтения файла с диска, процессор берет на выполнение другую

задачу. И возвращается к предыдущей при её готовности.

TPL Parallel – класс, предназначен для упрощения параллельного выполнения кода. Имеет ряд методов, которые позволяют распараллелить задачу. Реализует «настоящий» параллелизм, а не асинхронность как TPL Dataflow. Каждая задача, выполняется на отдельном потоке, полученном из пула потоков. В свою очередь, это создает дополнительные проблемы при разработке.

В результате было решено использовать TPL Dataflow и был реализован поток данных, показанный на рисунке 28.

В начале данного процесса расположен буфер, в который асинхронно считываются файлы из базы данных. Асинхронность данного процесса обусловлена задержками на обращение к базе данных и получения от нее ответа.

В это время каждый из блоков трансформации параллельно пытается получить данные (файл), сохраненный в буфере, если одному из блоков это удается он применяет к файлу действия, заложенные в себе.

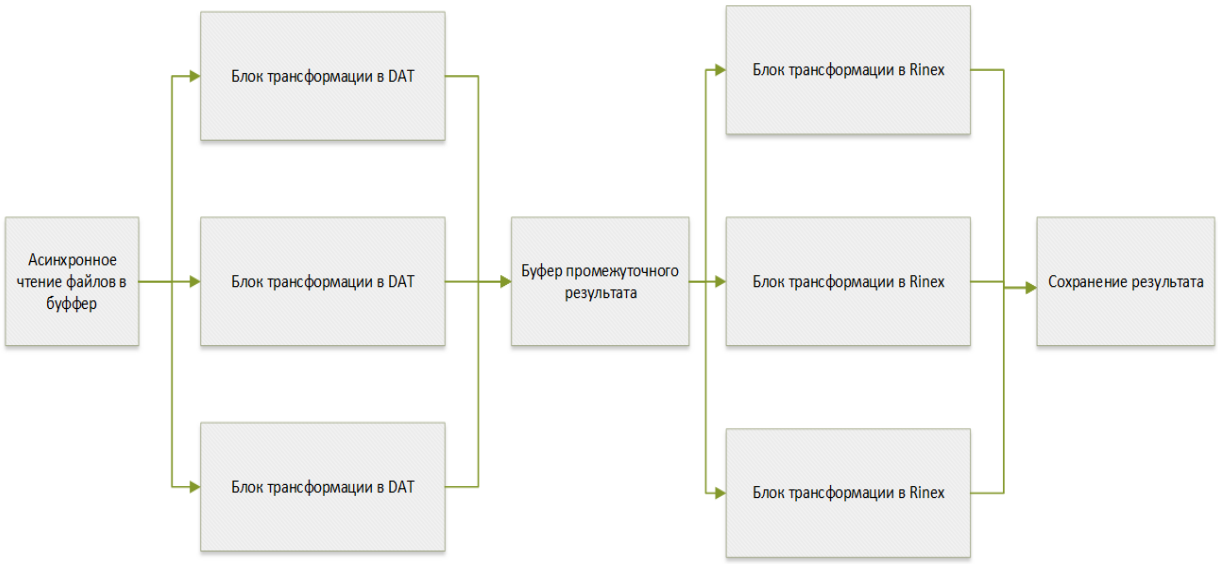

Рисунок 28 – Схема параллельной работы приложения конвертирования ГНСС-файлов

Данная операция продолжительна, поэтому если в буфере окажется ещё один файл, то его на обработку возьмет уже другой блок транформации. Количество блоков трансформации регулируется на этапе их настройки специаль-

ным свойством (рисунок 29).

Результат работы блоков трансформации в DAT складывается в следующий буфер, из которого, в свою очередь, пытаются получить данные блоки трансформации в Rinex-файл.

К преимуществам данного подхода можно отнести экономию процессорных ресурсов в момент простаивания блоков трансформации (в рамках операционной системы такой блок это просто не потребляющая ресурсов приостановленная задача, которая будет возобновлена, как только для нее появится работа в буфере). Так же при таком подходе, невозможно получить такие проблемы параллелизма как захват ресурса несколькими потоками, пропуска обработки ресурса.

```
var toDatTransformBlock = new TransformBlock<List<string>, List<string>>( transform: async filesInDays:List<string> => await TtoDat(filesInDays)
,new ExecutionDataflowBlockOptions
{
    MaxDegreeOfParallelism = 8
});
var toRinexActionBlock = new ActionBlock<List<string>>( action: async datFiles:List<string> => await DatToRinex(datFiles), new ExecutionDataflowBlockOptions
{
    MaxDegreeOfParallelism = 8
});
toDatTransformBlock.LinkTo(toRinexActionBlock);
```

Рисунок 29 – Блок трансформации в DAT

К недостаткам можно отнести меньшую скорость обработки чем при использовании класса Parallel. Это связано с разными механизмами действия данных библиотек. TPL Dataflow полагается на асинхронность операций (которая может происходить и в рамках одного потока), а Parallel в свою очередь гарантировано использует разные потоки процессора для выполнения работы.

3.1.4 Разработка приложения систематизации, сбора и хранения данных

Разработка приложения производится с помощью веб-фреймворка ASP NET Core. Входные и выходные данные приложения описаны на схеме функциональной модели микросервиса (рисунок 12). Структура программного решения показана на рисунке 30. В качестве СУБД, согласно проекту, используется MongoDB. База данных содержит коллекции, которые включают данные с

отдельных ГНСС-станций.

Рисунок 30 – Структура программного решения приложения

Приложение расположено в отдельном микросервисе. В рамках приложения обработки ГНСС-данных будет запущено два микросервиса с приложением систематизации, сбора и хранения данных. Один, для данных получаемых в рамках исследуемого региона, второй для данных, получаемых с серверов Национального Аэрокосмического Агентства (данные о траектории спутников, поправки, данные ионосферы и другая информация о состоянии спутников необходимая для расчетов).

Загрузка новых файлов со станций происходит асинхронно не нарушая других функций приложения. По окончании этого процесса и при условии отсутствия задачи в очереди брокера будет запущен процесс индексации. Процесс аналогичен касательно данных о спутниках.

При запуске процесса переиндексации приложение выполняет поиск добавленных или измененных с момента прошлой индексации файлов и запускает процесс генерирования метаданных.

Метаданными ГНСС-файлов являются: название станции, дата и время в трех форматах (классический, день года и ГНСС-дата). Данные метаданные в

дальнейшем позволяют осуществлять поиск файлов в любой из систем времяисчисления.

3.1.5 Разработка модулей приложения обработки ГНСС-данных

Одним из требований к компонентам приложения являлось независимость внутренней структуры от внешней среды. Для этого, используя подходы CQRS, Domain Driven Design и Onion, определим модули микросервиса (рисунок 31):

- модуль, содержащий основные классы и интерфейсы приложения GIS.GamitGlobk.Core.dll;

- модуль, обеспечивающий взаимодействие с программным комплексом Gamit/Globk – GIS.GamitGlobk.CommandShell.dll. Определяет набор программных-интерфейсов для вызова команд из исполняемых файлов пакета Gamit/ Globk;

- модуль с базовой логикой приложения GIS.GamitGlobk.Application.dll. Реализует CQRS подход для обработки входящих запросов;

- модуль, содержащий логику обработки запросов к API микросервиса GIS.GamitGlobk.Application.Api.dll. Реализует CQRS подход для обработки входящих запросов;

- модуль, содержащий точки HTTP-соединений (url-адреса) для клиентов микросервиса GIS.GamitGlobk.Api.

Для взаимодействия между внутренними частями системы используется паттерн Mediator, который сопоставляет CQRS вызовы со сборками которые будут его обрабатывать, гарантирует выполнение всех звеньев цепочки обработки и возвращает ответ сборке-источнику CQRS вызова.

Функциональная модель микросервиса обработки ГНСС-данных показана на рисунке 14. На вход, микросервис принимает команды и запросы от пользователя и ,используя правила маршрутизации и валидации передает их на обработку. Также данный микросервис реализует процедуру преобразования бинарных ГНСС-файлов в Rinex формат для чего использует конвертор реализованный в пункте 3.3.3. Сами файлы базовых станций (ГНСС-файлы) загружаются

асинхронно, после создания проекта, основываясь на периоде для проекта (не более 20 дней).Это позволяет создавать независимые структурные компоненты, менять и изменять их отдельно, несмотря на то что они совместно реализуют функционал приложения.

На рисунке 15 видно, что основная сборка GamitShellApi.dll реализует механизмы получения запросов и команд по веб-протоколу HTTP. После чего команда/запрос передается посреднику, который находит соответствующего команде/запросу обработчика. Но перед выполнением инструкций обработчика произойдет применение зарегистрированных для той или иной команды/запроса проверок.

В случае, если входные данные содержат ошибку цепочка обработки (pipeline) будет разрушена и произойдет выброс ошибки с указанием какие условия для каких параметров были нарушены.

В результате разработки были получены зависимости между модулями отраженные на рисунке 31.

Внутренняя структура приложения микросервиса построена по принципу CQRS (Command Query Responsibility Segregation (разделение ответственности на команды и запросы)). В рамках данного подхода всё общение с логикой приложения с троится на основе запросов (Query) и команд (Command). Команда используется для изменения объектов (операции записи, например запись в базу данных). Для получения какой-либо информации используется запрос (query). Такой подход позволяет обеспечить оптимальное время ответа даже при большой нагрузке на одну из подсистем. Например, при сборе ГНСС-данных часть системы ответственная за хранение данных будет нагружена, но благодаря CQRS подходу пользователь будет все также быстро получать ответ от подсистемы чтения данных.

DDD (Domain Driven Design) [58] – подход, который предлагает в рамках программного кода использовать терминологию предметной области. Это упрощает взаимодействие с заказчиком, так как позволяет разговаривать программисту и заказчику на языке предметной области.

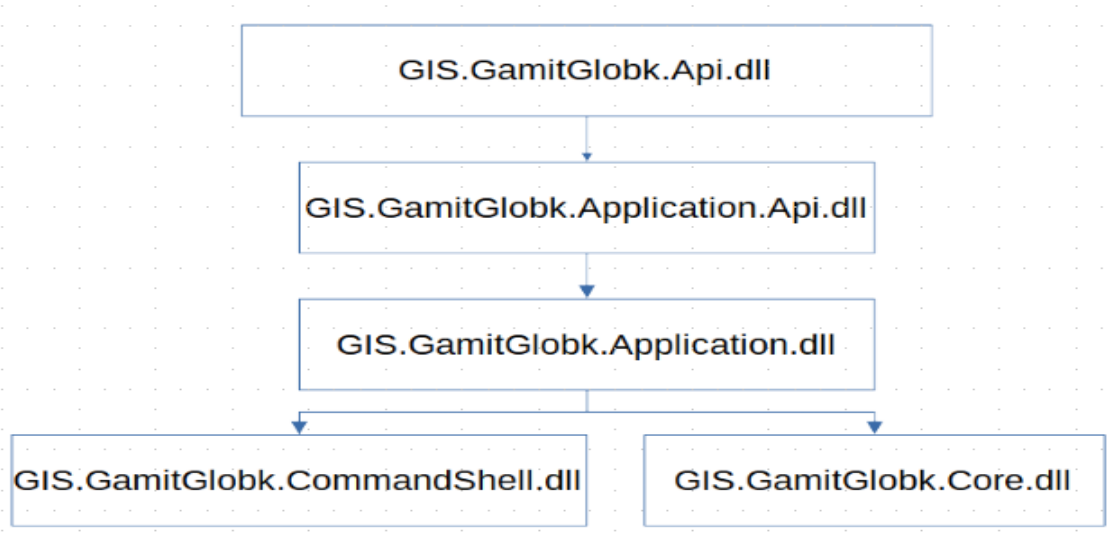

Рисунок 31 – Схема зависимостей модулей микросервиса

Согласно архитектурному паттерну «Onion» зависимости между слоями должны распространяться только сверху в низ (рисунок 31), также допускаются взаимодействия между слоями одного уровня.

Модуль GIS.GamitGlobk.Core

Модуль представляет из себя библиотеку классов netstandard. Содержит в себе следующие пространства имен: Entities, Extensions, Interfaces, Models. Они содержат в себе элементы (классы и интерфейсы, расширения и сущности), которые используются во всех вышестоящих модулях (рисунок 32).

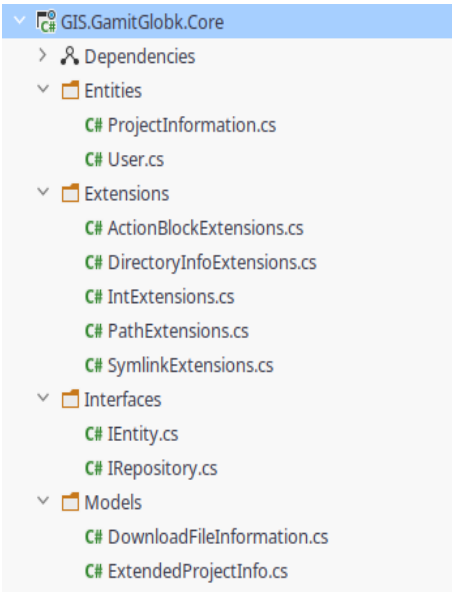

Рисунок 32 – Структура модуля GIS.GamitGlobk.Core

Так, например, SymlinkExtensions реализует функционал создания символических ссылок в файловой системе. Этот механизм позволяет не перемещать большие объемы данных при создании пользователем проекта и иметь локальный кеш этих данных (когда пользователь создает проект, ссылки на нужные файлы помещаются в проект, тем самым не производится операция копирования данных). Интерфейсы IEntity и IRepository реализуют контракты сущности СУБД и источника данных соответственно.

Модуль GIS.GamitGlobk.CommandShell

Модуль представляет из себя библиотеку классов netstandard. Содержит объекты (классы) для взаимодействия с математическим комплексом GamitGlobk. Создание данного модуля обуславливается необходимостью взаимодействия с математическим пакетом из современных объектно ориентированных языков. В результате создания данного **м**одуля обращение к математическому пакету станет более безопасным, а так же в модулях верхнего уровня будут доступны механизмы обработки исключительных ситуаций. ShCommand содержит объектно ориентированные обертки для команд GamitGlobk. Models – содержит аргументы для этих команд. Структура проекта приведена на рисунке 33.

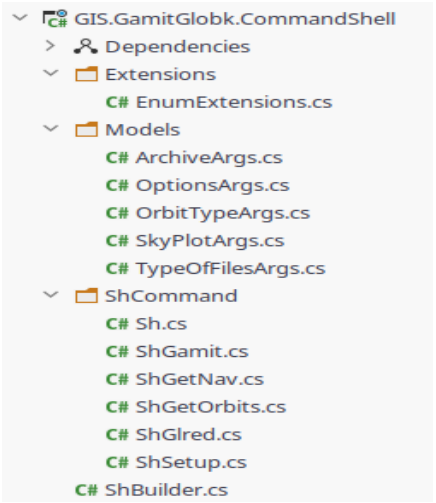

Рисунок 33 – Структура модуля GIS.GamitGlobk.CommandShell

Цель создания данного модуля – обеспечить объектно-ориентированный

доступ к комплексу GamitGlobk из платформы dotnet. Например, OrbitTypeArgs представляет из себя тип enum (рисунок 34) с набором элементов. Специальный атрибут [Flags] и сдвиг каждого элемента относительно предыдущего позволяет комбинировать входные параметры (рисунок 35).

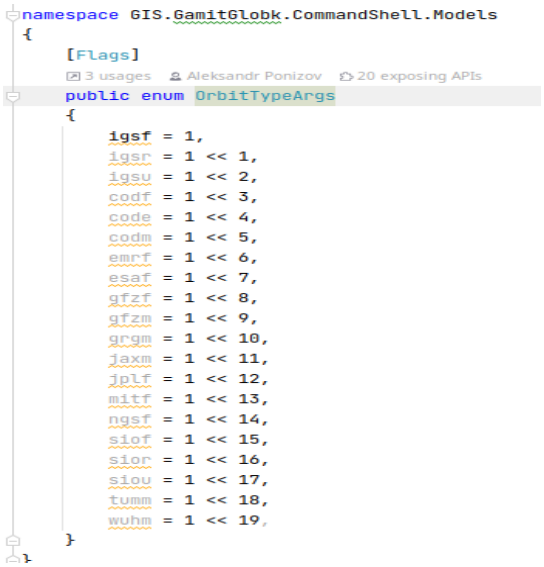

Рисунок 34 – Реализация OrbitTypeArgs

Так, например для комбинирования флагов igsf и igsr можно использовать приведенную ниже конструкцию.

```
arg.Add("-orbit").Add(OrbitTypeArgs.igsf & OrbitTypeArgs.igsr)
```

Рисунок 35 – Использование OrbitTypeArgs

Данный код будет преобразован компилятором в логическую операцию И между двумя элементами: igsf (01) И igsr (10) = 11. Это позволит передавать несколько параметров явным образом. Так же, данное решение помогает следовать подходу DDD, ведь заказчик (человек, работавший с пакетом GamitGGlobk) посмотрев на данный код увидит знакомые ему термены предметной области (рисунок 35).

Модуль GIS.GamitGlobk.Application

Данный модуль представляет собой библиотеку классов netcore3.1. Она объединяет в себе функционал, общий для всех проектов из пространства имен

GIS. * (рисунок 36). Также является точкой начальной конфигурации всех приложений. Для этого она реализует класс Configure.cs (рисунок 37).

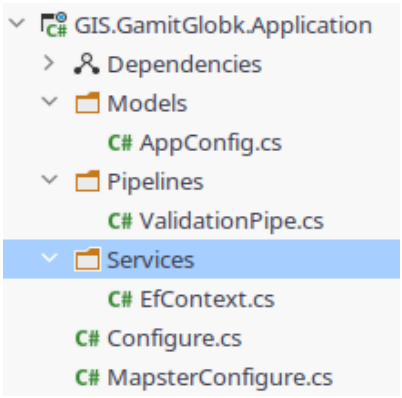

Рисунок 36 – Структура модуля GIS.GamitGlobk.Application

Класс Configure.cs является статическим (так как представляет собой точку входа для модулей верхнего уровня) и содержит следующие методы: AddGamitApplication(…), WithMediatorToAssemblies(…), UseMapsterRules. Пример конфигурации в рамках модульного тестирования приведен на рисунке 37.

```
1 usage  Aleksandr Ponizov +1 *
public static class Configure
{
    public static (IServiceCollection services, AppConfig appConfig)
        4 usages  Aleksandr Ponizov +1 *
        AddGamitApplication(this IServiceCollection services, Action<AppConfig> configureApp)
    {
        var config = new AppConfig();
        configureApp(config);
        services.AddSingleton(config);

        services.AddDbContext<EfContext>( optionsAction: options
            => options.UseMySql(config.DataBaseConnection), ServiceLifetime.Transient);
        return (services, config);
    }

    public static (IServiceCollection services, AppConfig configureApp)
        4 usages  Александр Понизов +1 *
        WithMediatorToAssemblies(this (IServiceCollection services, AppConfig configureApp) servicesConfig,
            params Assembly[] assembly)
    {
        var (services :IServiceCollection , app) = servicesConfig;
        var currentAsm :Assembly  = typeof(Configure).Assembly;
        var includedAsm :Assembly[]  = assembly.Append(currentAsm).ToArray();

        services.AddMediatR(includedAsm);
        services.AddValidatorsFromAssemblies(includedAsm);
        services.AddTransient( serviceType: typeof(IPipelineBehavior<,>),  implementationType: typeof(ValidationPipe<,>));

        return servicesConfig;
    }

    1 usage  Aleksandr Ponizov
    public static void UseMapsterRules(this IApplicationBuilder app) => MapsterConfigure.RegisterCustomRules();
}
```

Рисунок 37 – Класс Configure.cs

Метод AddGamitApplication инициализирует базовый для приложений функционал (чтение конфигурационного файла в объект, создание подключение к базе данных). Метод WithMediatorToAssembly() принимает на вход указатели на сборки (dll) и добавляет их в отслеживаемые для «посредника».

```
private static ServiceProvider SetUpServiceProvider()
{
    var services = new ServiceCollection();

    services.AddGamitApplication( configureApp: s :AppConfig => FillTestAppConfig(ref s))
            .WithMediatorToAssemblies(GamitGlobk.Application.Api.Configure.GetApiAssembly())
            .WithApi();

    MapsterConfigure.RegisterCustomRules();

    return services.BuildServiceProvider();
}
```

Рисунок 38 – Конфигурация окружения для модульного тестирования

Это позволяет «посреднику» перенаправлять входящие запросы и команды между разными модулями микросервиса. При этом сохраняется принцип инкапсуляции: все модули знают о посреднике, но ничего не знают друг о друге.

Модуль GIS.GamitGlobk.Application.Api

Данный модуль содержит специфичный код, используемый только микросервисом обработки ГНСС-данных. На языке терминов подхода DDD такой код называются «бизнес-логикой». Структура проекта показана на рисунке 39.

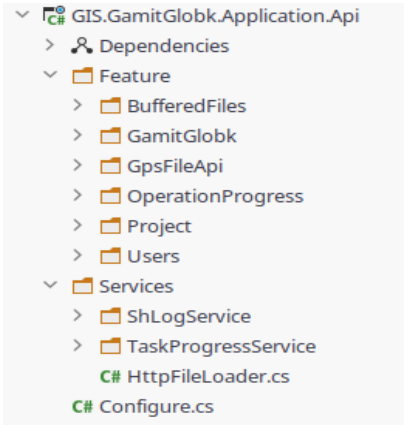

Рисунок 39 – Структура модуля GIS.GamitGlobk.Application.Api

Функции расположены в пространстве имен Feature и реализуются по-

средством описания команды или запроса, «валидатора» и обработчика (рисунок 40).

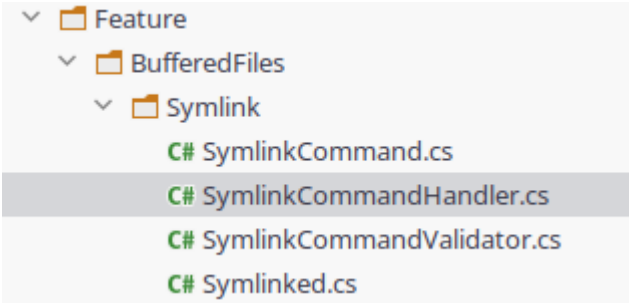

Рисунок 40 – Структура функции создания символических ссылок

На рисунке 40 представлены следующие элементы:

- SymlinkCommand.cs – команда на создание символических ссылок;
- SymlinkCommandValidator.cs – «валидатор», который выполняет проверку данных, содержащихся в команде, в случае провала возвращает ошибку, тем самым прерывая цепочку обработки;
- SymlinkCommandHandler.cs – обработчик команды;
- Symlinked.cs – действие, которое выполняется при успешном завершении работы обработчика (например, логирование результата).

Модуль GIS.GamitGlobk.Api

Модуль верхнего уровня, представляет из себя исполняемый файл приложения. В нем выполняется конфигурация всех сервисов и служб необходимых микросервису обработки ГНСС-данных. Также именно здесь реализуется создание точек HTTP соединения (url). Для этого создаются контроллеры («Controllers»). Контроллеры могут содержать в себе различное количество методов get, post, put. Именно с этим приложением взаимодействуют другие микросервисы системы. Принцип работы приложения такой:

- приходит HTTP запрос от клиента;
- проверяется наличие подходящей конечной точки (например, /api/symlink);
- если точка найдена, то входящий запрос передается в подходящий метод

контроллера;

- если не найдена, то возвращается ошибка 404 (точка не найдена).

В результате разработки было получено приложение для микросервиса обработки ГНСС-данных, соответствующий спецификации и архитектурному проекту. Данное приложение обладает всеми необходимыми качествами: слабая связанность внутренних модулей, легковесность (после запуска микросервис потребляет порядка сорока пяти мегабайт оперативной памяти), легкость в обслуживании и модернизации.

3.1.6 Разработка графического интерфейса пользователя

Графический интерфейс представлен веб-приложением, построенным на базе фреймворка Blazor и набора компонентов Material Blazor.

Blazor позволяет создавать интерактивные веб-интерфейсы с использованием C# вместо JavaScript. Приложения Blazor состоят из повторно используемых компонентов веб-интерфейса, реализованных с использованием C#, HTML и CSS. И клиентский, и серверный код написан на C#, что позволяет вам совместно использовать существующий код и библиотеки.

Blazor может запускать клиентский код C# прямо в браузере, используя WebAssembly. Поскольку это настоящий .NET, работающий на WebAssembly, вы можете повторно использовать код и библиотеки из серверных частей вашего приложения.

В качестве альтернативы Blazor может запускать вашу клиентскую логику на сервере. События пользовательского интерфейса клиента отправляются обратно на сервер с помощью SignalR - инфраструктуры обмена сообщениями в реальном времени. После завершения выполнения необходимые изменения пользовательского интерфейса отправляются клиенту и объединяются в DOM.

Так как планируется использовать приложение в границах локальной сети предприятия, было выбрано серверное размещение приложения, это позволит разгрузить устройства пользователей и проводить все необходимые вычисления UI на стороне сервера.

3.2 Компиляция и создание образов микросервиса

Микросервисы – это архитектурный подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP.

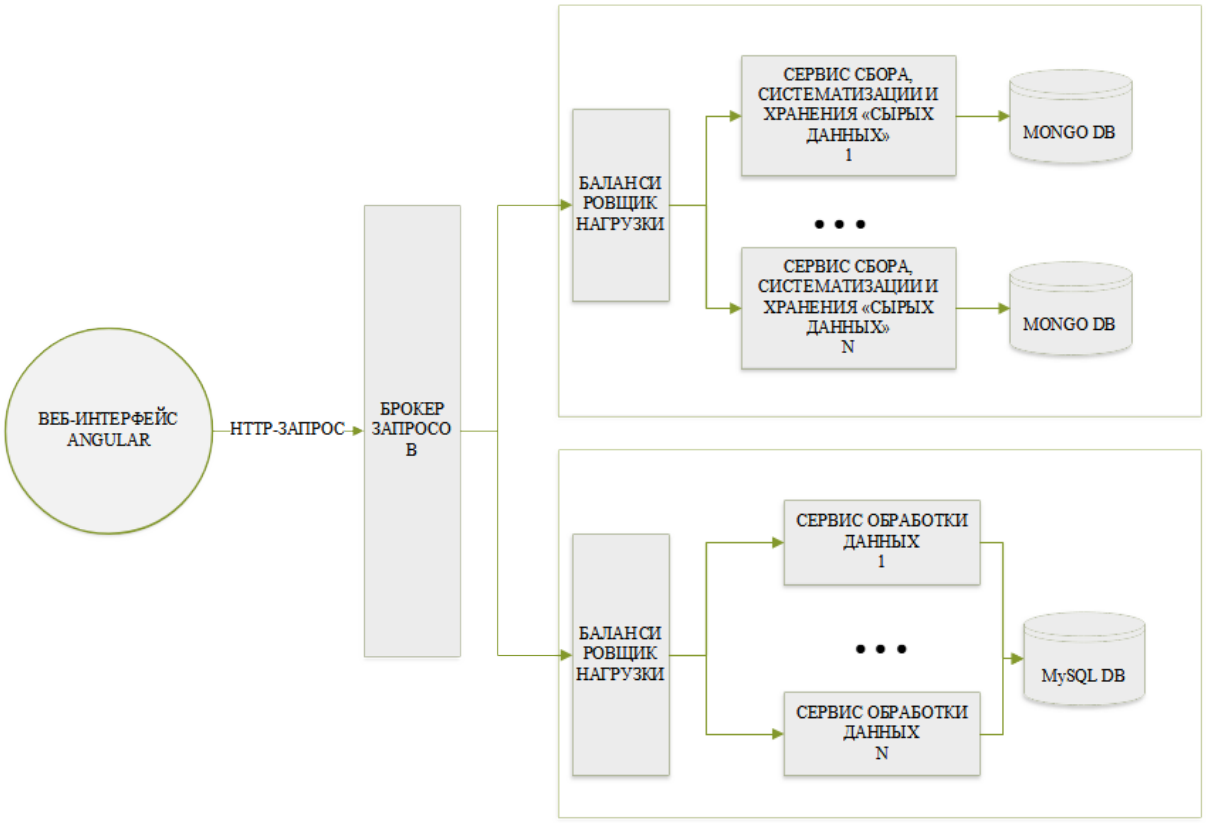

Рисунок 41 – Схема масштабирования микросервисов обработки данных

Эти сервисы построены вокруг бизнес-потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

На рисунке 41 видно, что распределение запросов между развернутыми микросервисами происходит с помощью брокера запросов. Брокер запросов использует понятие очередей и подписчиков. Очередь это структура в памяти брокера, которая хранит клиента и полученное от него сообщение. Это сообщение

будет забрано первым свободным подписчиком конкретной очереди.

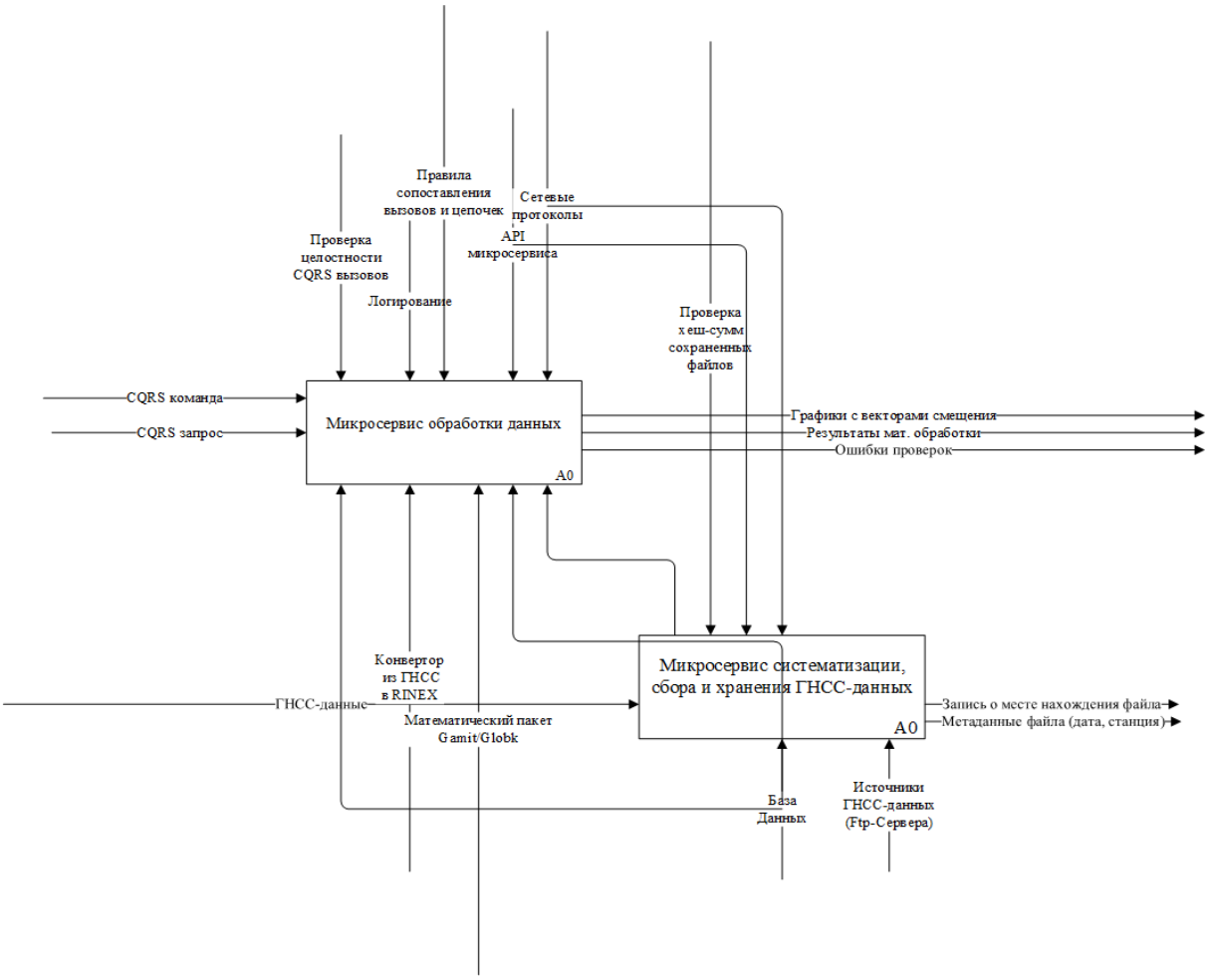

Рисунок 42 – Схема взаимодействия микросервисов приложения

В случаях когда необходимо чтобы подписчик обрабатывал все запросы от одного клиента (обслуживал его) возможно сделать дополнительную настройку при сооздании очереди. Схема с использованием брокера запросов очень удобна, так как позволяет беспроблемно горизонтально масштабировать систему в критически важных узлах.

Схема взаимодействия микросервисов приложения обработки ГНСС-данных показана на рисунке 42.

Для увеличения производительности, в рамках приложения применяется горизонтальное масштабирование, то есть запуск нескольких экземпляров одновременно (рисунок 41). Благодаря использованию микросервисного архитектурного подхода, дальнейшая модернизация приложения может производиться

за счет написания новых микросервисов на любых языках и платформах, поддерживающих взаимодействие по HTTP протоколу.

Микросервисы эксплуатируются в виде так называемых образов. Образ представляет собой приложение со всеми зависимостями (в данном случае зависимостями также считается зависимость от компонентов операционной системы (кроме ядра ОС)). Для создания образа в экосистеме Docker необходимо создать конфигурационный файл «dockerfile». Внешний вид которого для микросервиса обработки данных показан на рисунке 43.

```
# Build FDOApiApp
FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /source
COPY *.sln .

# Start copy csproj for restoring
COPY GIS.Api/*.csproj ./GIS.Api/
COPY GIS.Api.Tests/*.csproj ./GIS.Api.Tests/
COPY GIS.Application/*.csproj ./GIS.Application/
COPY GIS.Application.Api/*.csproj ./GIS.Application.Api/
COPY GIS.Core/*.csproj ./GIS.Core/
COPY GIS.GamitGlobk.CommandShell/*.csproj ./GIS.GamitGlobk.CommandShell/
COPY GIS.GamitGlobk.CommandShell.Tests/*.csproj ./GIS.GamitGlobk.CommandShell.Tests/
COPY GIS.UI/*.csproj ./GIS.UI/
COPY GIS.UI.Application/*.csproj ./GIS.UI.Application/
# Next line fix docker bug
RUN true
COPY GIS.UI.Tests/*.csproj ./GIS.UI.Tests/
RUN dotnet restore
# copy everything else and build app
COPY GIS.Api/. ./GIS.Api/
COPY GIS.Api.Tests/. ./GIS.Api.Tests/
COPY GIS.Application/. ./GIS.Application/
COPY GIS.Application.Api/. ./GIS.Application.Api/
COPY GIS.Core/. ./GIS.Core/
COPY GIS.GamitGlobk.CommandShell/. ./GIS.GamitGlobk.CommandShell/
COPY GIS.GamitGlobk.CommandShell.Tests/. ./GIS.GamitGlobk.CommandShell.Tests/
COPY GIS.UI/. ./GIS.UI/
COPY GIS.UI.Application/. ./GIS.UI.Application/
WORKDIR /source
RUN dotnet build -c Release
# End build

# Publish api app
FROM build AS publishApi
WORKDIR /source/GIS.Api
RUN dotnet publish -c Release -p:PublishReadyToRunUseCrossgen2=True -o /api_app --no-restore
# <--publishApi--> /api_app
```

Рисунок 43 – Dockerfile микросервиса обработки данных

Как правило dockerfile состоит из нескольких частей-этапов. Первый этап – это восстановление зависимостей, компиляция приложения, публикация и создание образа. Такой процесс создания докер-образов называется многостадийным. Каждый шаг (строка инструкции в dockerfile) получает уникальный в рамках процесса идентификатор. Преимуществом многостадийного процесса создания контейнеров перед одностадийным заключается в кешировании проме-

жуточных шагов. Кеширование шагов процесса экономит время и место на жестком диске. Многостадийный процесс сборки докер файла показан на рисунке 44. Запуск сборки образа происходит командой docker build.

На рисунке видно, что в начале каждой стадии необходимо выбрать основу, на базе которой будет создан образ. Как правило, за основу берутся образы операционных систем Debian, Alpine и другие.

```
Step 33/38 : FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS base
 ---> 75ad31c583c9
Step 34/38 : FROM base AS finalApi
 ---> 75ad31c583c9
Step 35/38 : WORKDIR /app
 ---> Using cache
 ---> 8a6cea22a792
Step 36/38 : COPY --from=publishApi /api_app .
 ---> e0c7b951241c
Step 37/38 : RUN mkdir wwwroot
 ---> Running in efdb633325b3
Removing intermediate container efdb633325b3
 ---> 385abcaacd37
Step 38/38 : ENTRYPOINT ["dotnet", "GIS.Api.dll"]
 ---> Running in 552e1505327e
```

Рисунок 44 – Лог сборки образа микросервиса обработки данных (финальная стадия)

Эти образы, в отличие от одноименных операционных систем не содержат ядра и приложений: управления системными ресурсам; помощи пользователю. Это позволяет довольно сильно уменьшить размер образа. Так образ Alpine Linux весит всего лишь 5.61 Мб.

После создания образов для микросервисов: сбора и хранения информации, обработки информации и пользовательского интерфейса необходимо регламентировать отношения между ними и их совместный запуск и закрытие. Так же для нужд приложения при его запуске должна запускаться СУБД MySQL, MongoDB и Apache 2 в качестве обратного прокси сервера. Для того чтобы оформить набор образов как единое приложение с описанными внутренними связями возможно использовать три нструмента: docker compose, docker swarm или kubernetes. Среди этих трех инструментов только Docker-compose не позволяет управлять контейнерами на разных хост машинах. Связывание приложний

вместе происходит посредством написания docker-compose.yml файла (рисунок 45)

Структурно docker-compose файл состоит из нескольких секций. Основная секция «services» содержит описание создаваемых микросервисов. Например, на рисунке видно, что микросервис gisapiapp в качестве образа использует dockerfile (а не готовый образ), и при его создании будут проброшены порт 80 с контейнера на порт 5000 хоста.

```
version: '3.4'
services:
  gisapiapp:
    container_name: gisapiapp
    mem_limit: 300m
    ports:
      - "127.0.0.1:5000:80"
    build:
      context: ./
      dockerfile: ./Dockerfile
      target: finalApi
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
    restart: always
  gisuiapp:
    container_name: gisuiapp
    mem_limit: 300m
    ports:
      - "127.0.0.1:5005:80"
    build:
      context: ./
```

Рисунок 45 – Docker-compose файл приложения обработки ГНСС-данных

Подобным образом описываются все микросервисы. Для запуска приложения используется команда docker-compose up d. Для остановки docker-compose down.

3.3 Выполнение тестирования и отладки программного продукта

Для обеспечения непрерывного цикла разработки в рамках подхода «экстремального программирования» и выбранного ранее жизненного цикла «спиральная модель» необходимо обеспечить покрытия тестами исходных кодов.

В качестве выбранной методики написания тестов была выбрана Test-

driven Development (разработка через тестирование) [17]. Данная методология основывается на повторении коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти этот тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.

В результате разработки микросервиса обработки ГНСС-данных было сформировано два набора тестов: для проекта GIS.GamitGlobk.Api.Tests (рисунок 46) и GIS.GamitGlobk.CommandShell.Tests (рисунок 47).

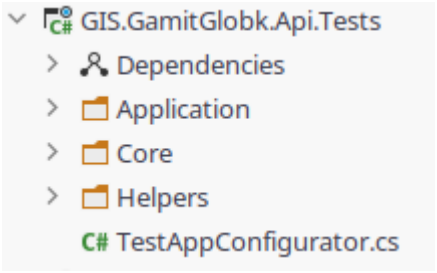

Рисунок 46 – Структура проекта тестирования GIS.GamitGlobk.Api.Tests

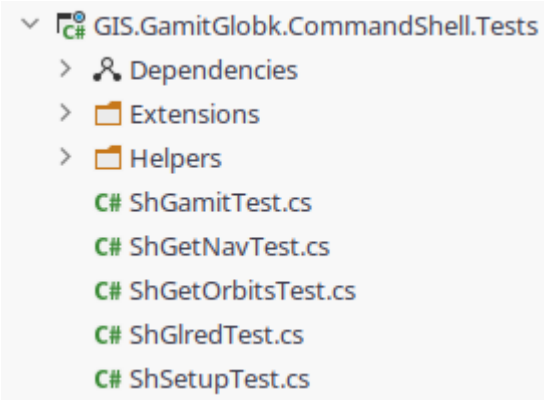

Рисунок 47 – Структура проекта тестирования GIS.GamitGlobk.CommandShell.Tests

Для микросервиса сбора и хранения был разработан набор тестов GIS.GpsFile.WebApi.Tests. Результаты прогона тестов показаны на рисунке 48. Площадь покрытия проекта тестами показана на рисунках 49. Так же тестами покрывались все компоненты и доп модули используемые внутри микросервисов.

Тестирование – неотъемлемая часть процесса современной разработки программного обеспечения. На начальных этапах, когда необходимо быстро создать прототип программного обеспечения не стоит применять тестирования.

Но в дальнейшем, тесты – это важный инструмент, позволяющий программисту изменять код в любом месте системы в соответствии с изменяющимися требованиями.

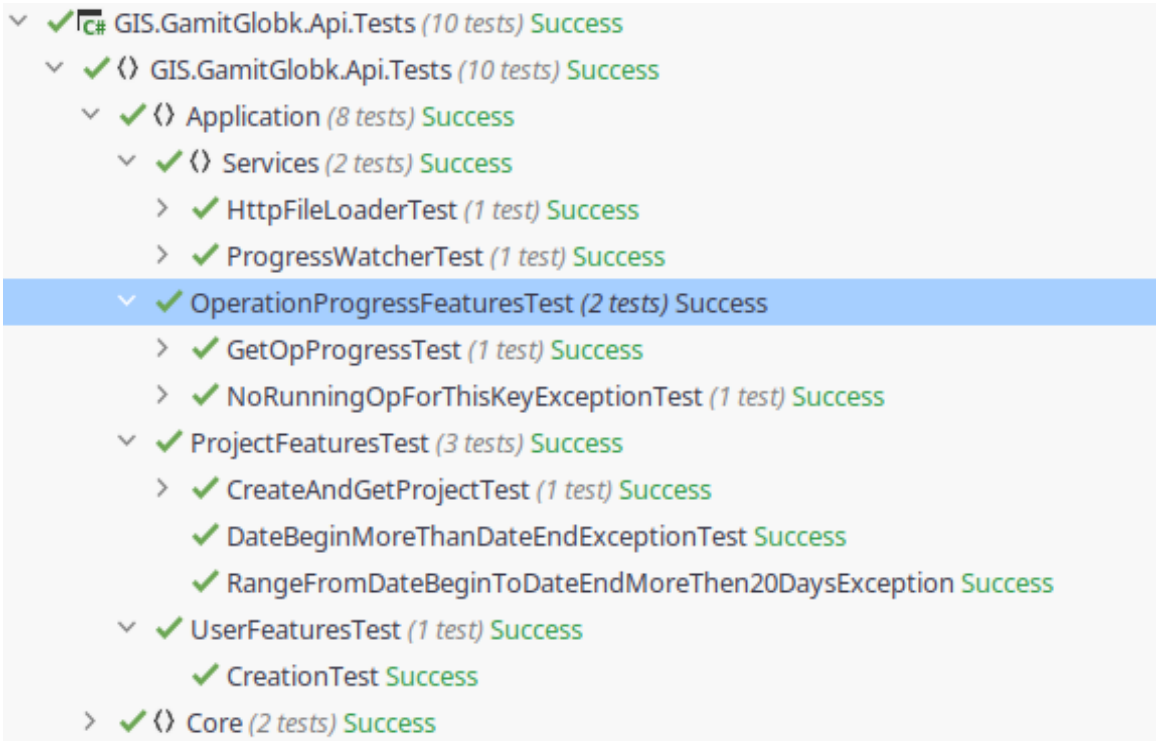

Рисунок 48 – Результат прохождения тестов

На рисунке видно, что тесты завершились успешно. Однако, так бывает не всегда. В случае, если тесты завершаются ошибкой, программист может использовать инструменты отладки на тестовом примере и быстро найти и устранить проблему.

Если бы тестов не было, вносить изменения в проект становилось бы сложнее с каждым витком жизненного цикла. Множество скрытых ошибок выявлялись бы только при эксплуатации продукта.

Площадь покрытия решения тестами это качественная характеристика, которая показывает количество протестированного кода и не протестированно-

го.

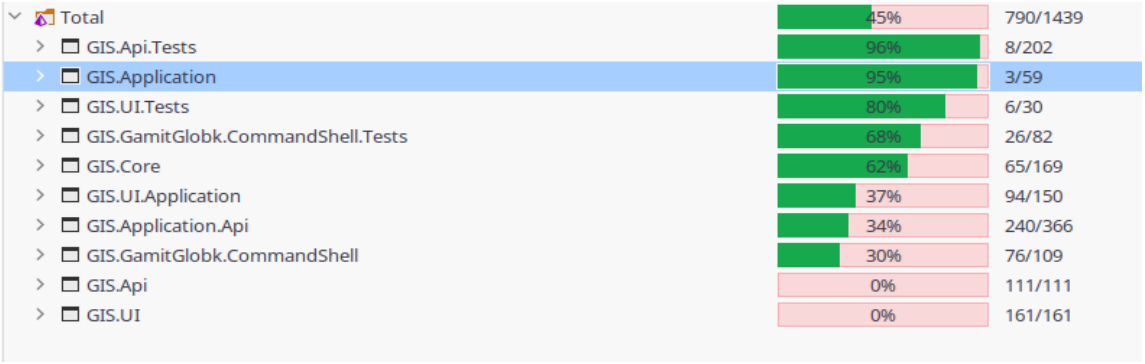

Рисунок 49 – Площадь покрытия теста для решения микросервиса обработки ГНСС-данных

То есть проект, где 60 % кода покрыто тестами имеет 40 % кода, который вообще не затрагивается тестами. На рисунке 26 мы видим, что решение, содержащее проекты с объектной оберткой над GamitGlobk, обработки ГНСС-файлов и графического интерфейса, суммарно имеет 45 % покрытия тестами.

Одними из самых протестированных модулей (за исключением тестов) являются GIS.Application(95 %), GIS.Core (68 %), GIS.UI.Application(37 %), GIS.Application.Api(34 %), GIS.GamitGlobk.CommandShell(30 %). Эти решения содержат основную бизнес-логику, которая решает поставленные перед приложением задачи. Наименее же протестированные части – это приложения самого верхнего уровня, которые получают HTTP-запросы и делегируют полномочия нижележащим слоям.

3.4 Анализ достоверности и практической значимости результата

Подготовка программно-аппаратного комплекса

Для запуска приложения необходимо подготовить программную среду. В качестве программной среды используется платформа Proxmox Virtual Environment

Создание контейнера

В начале необходимо создать контейнер, где будут запущены микросервисы. Для этого нажмем кнопку «Создать CT» (рисунок 50), после чего откроется

меню, показанное на рисунке 51.

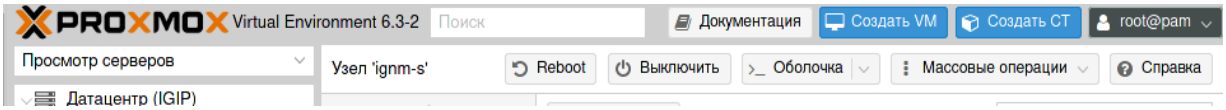

Рисунок 50 – Меню опций датацентра

На первом этапе заполняются данные о будущем контейнере: имя контейнера; пул ресурсов; пароль. Пул ресурсов – группа, к которой должен относиться создаваемый контейнер.

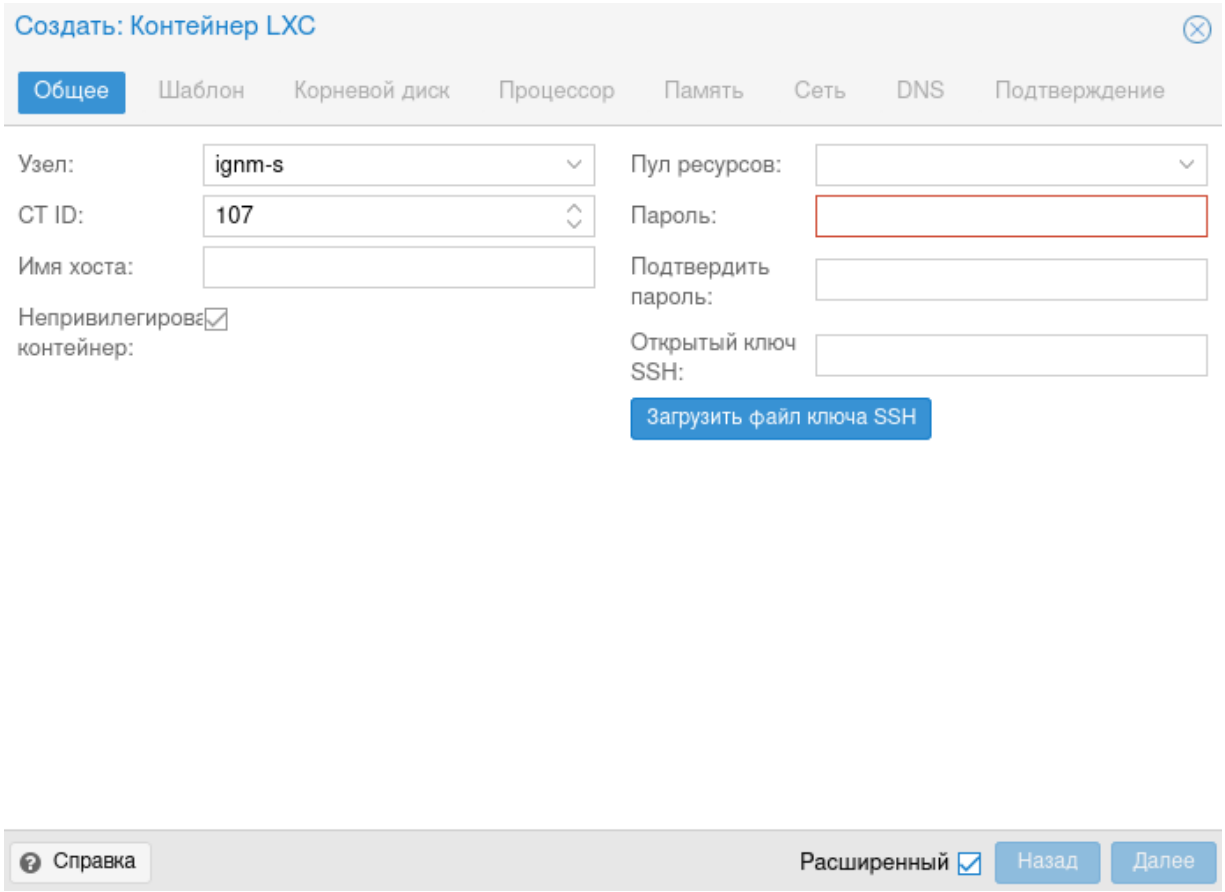

Рисунок 51 – Первый этап создания контейнера

Далее необходимо выбрать шаблон контейнера и его расположение. Шаблоном является образ операционной системы, на базе которой будет построен контейнер.

На рисунке 52 требуется выбрать расположение будущего контейнера и размер доступного ему пространства. В дальнейшем это значение можно изме-

нить «на лету».

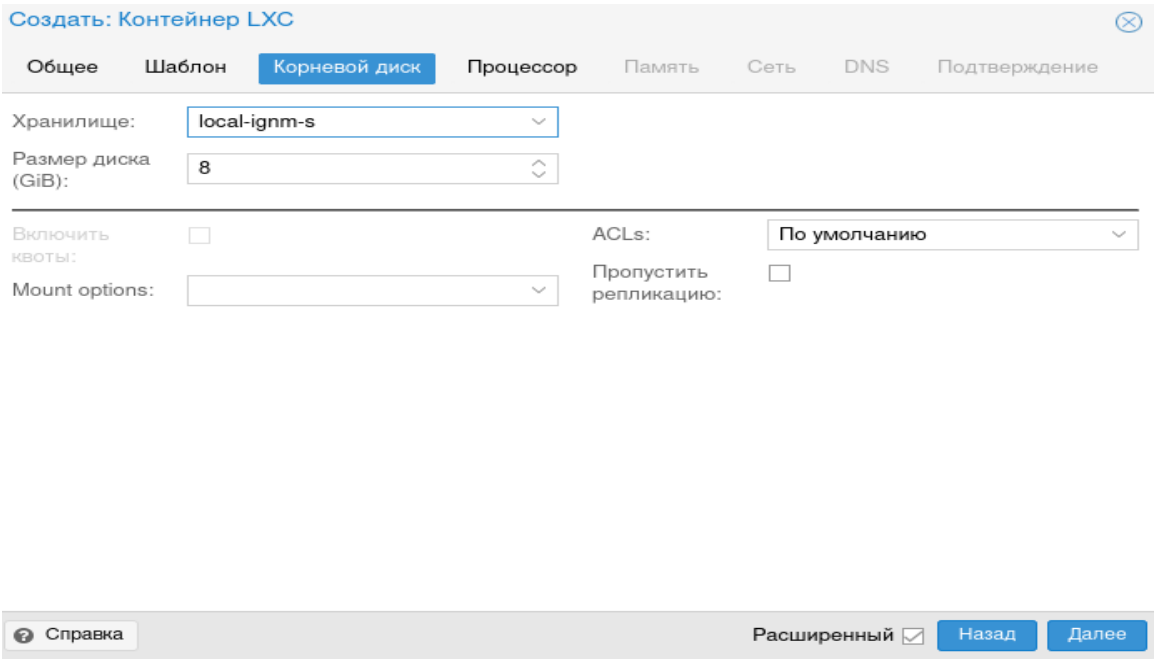

Рисунок 52 – Выбираем место расположения виртуального хранилища и его размер

Далее выделяются ресурсы процессора и оперативной памяти для контейнера. Настроим сетевой интерфейс контейнера (рисунок 53).

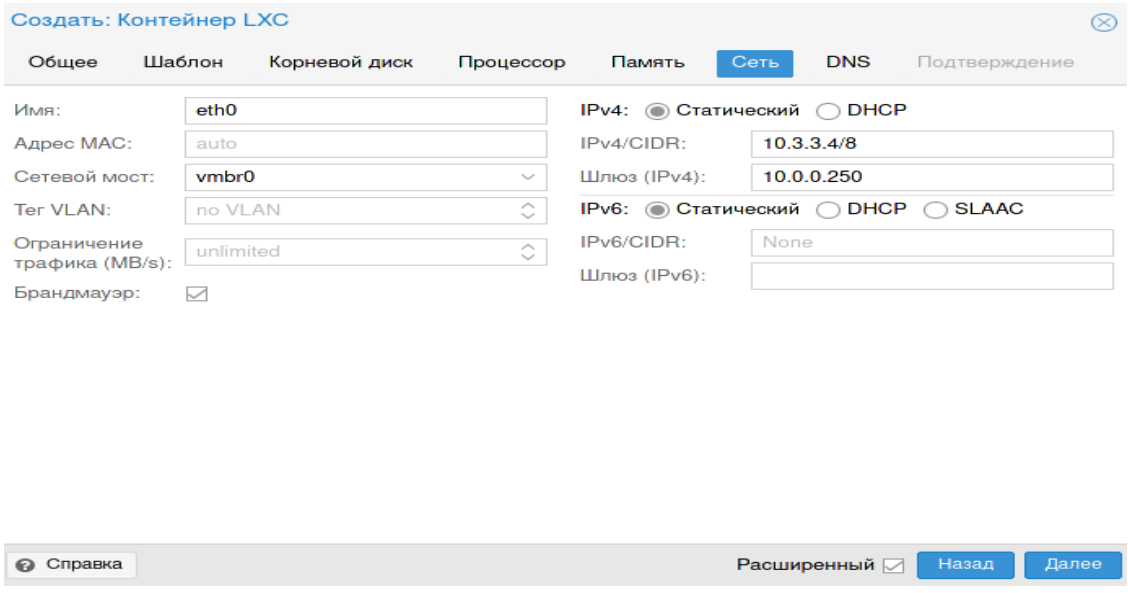

Рисунок 53 – Настройка сетевого интерфейса контейнера

При настройке сетевого интерфейса важно заполнить поля ip адреса и

шлюза. Поле ip адреса заполняется в формате ip-адрес/маска.

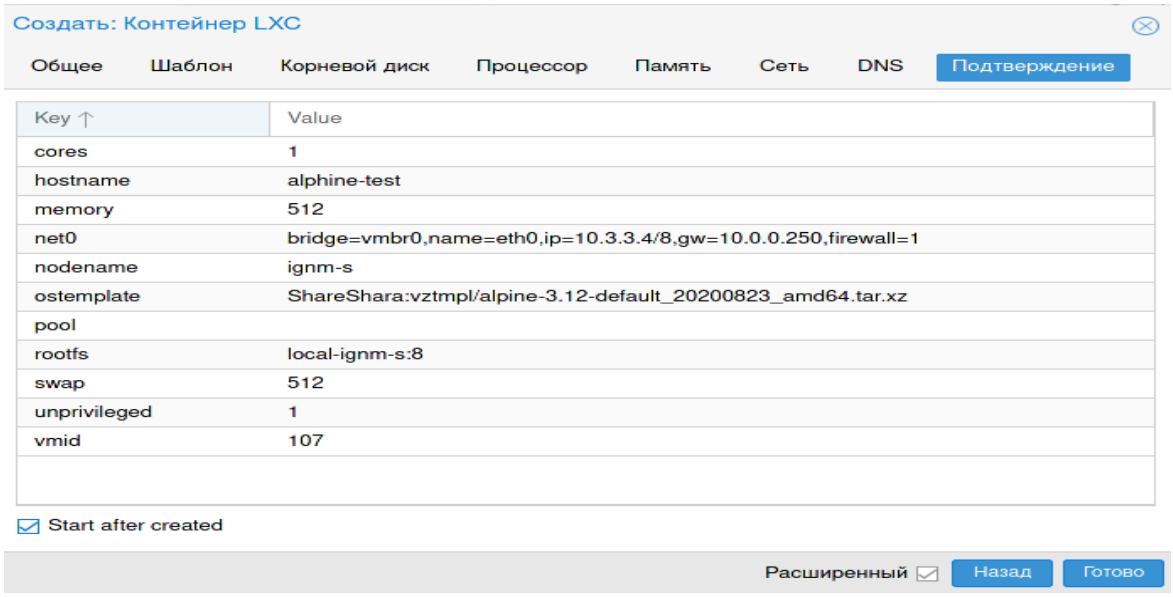

Рисунок 54 – Проверим настройки

На рисунке 54 изображен блок «Подтверждение», который содержит информацию обо всех сделанных нами ранее настройках.

Публикуем приложение

Опубликуем приложение .net core как «single-file» и R2R, для этого используем команду:

dotnet publish -r linux-x64 -p:PublishSingleFile=true -p:PublishReadyToRun=true

Результат выполнения команды изображен на рисунке 55.

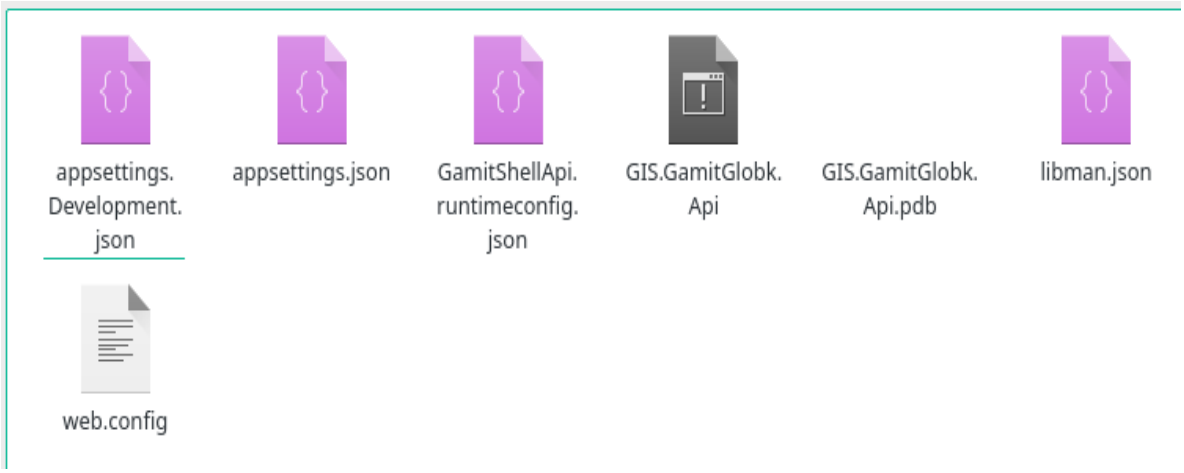

Рисунок 55 – Результат публикации приложения

В результате получим приложение, для использования которого не требуется устанавливать зависимости .net core 3.1.

Теперь скопируем файлы в контейнер и создадим конфигурацию службы для systemd (рисунок 56). Это позволит отслеживать состояние нашего приложения внутри контейнера и автоматически перезапускать его при необходимости.

```
[Unit]
Description= GpsFileInfo .Net Web API APP running on Ubuntu

[Services]
WorkingDirectory=/home/alex/gpsFileApiApp
ExecStart=/home/alex/gpsFileApiApp/GpsFileWebApi
Restart=always
RestartSec=10
SyslogIdentifier=dotnet-gps
User=root
Environment=ASPNETCORE_ENVIRONMENT=Production

[Install]
WantedBy=multi-user.target
```

Рисунок 56 – Пример конфигурации systemd службы

Включим службу командой:

sudo systemctl enable gamitApiApp.service

Просмотреть статус службы можно командой (рисунок 57):

sudo systemctl status gamitApiApp.service

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://[::]:5005
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Portable\RiderProjects\gamitglobkwebapi\GIS.Api
```

Рисунок 57 – Статус работающего приложения внутри контейнера

В результате получаем приложение, запущенное в контейнере. Создав ша-

блон этого контейнера, получим возможность быстро развертывать экземпляры нашего микросервиса.

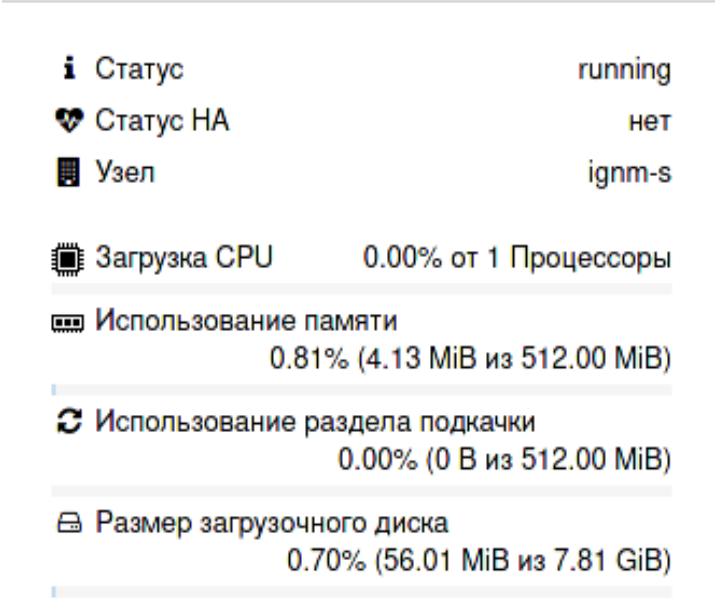

Рисунок 58 – Статус запущенного микросервиса

Микросервисы (маленькие приложения в контейнерах) позволяют быстро создавать инфраструктуру информационной системы (ГИС), являются легко заменяемыми, быстро развертываемыми и не накладывают дополнительные расходы на аппаратную часть сервера.

3.5 Достоверность и практическая значимость результатов

После авторизации, пользователю загружаются все его проекты. Для создания проекта пользователю достаточно нажать на кнопку «+» вверху страницы, ввести имя проекта и выбрать период для расчетов. Период ограничен 20 днями. В случаях, когда пользователь уверен, что настройки по умолчанию не подойдут, он может выбрать функцию расширенные настройки и после этого более точно настроить процесс.

После создания проекта (рисунок 59) приложение покажет пользователю уведомления о успешно поделанных действиях (рисунок 60) (создание файловой структуры проекта, записи в базе данных, инициализации проекта Gamit-Globk). Такие элементы делают интерфейс пользователя более отзывчивым и дружелюбным. В случае если пользователь не закрыл уведомления, они поме-

щаются в раздел непрочитанных и он сможет ознакомиться с ними позднее.

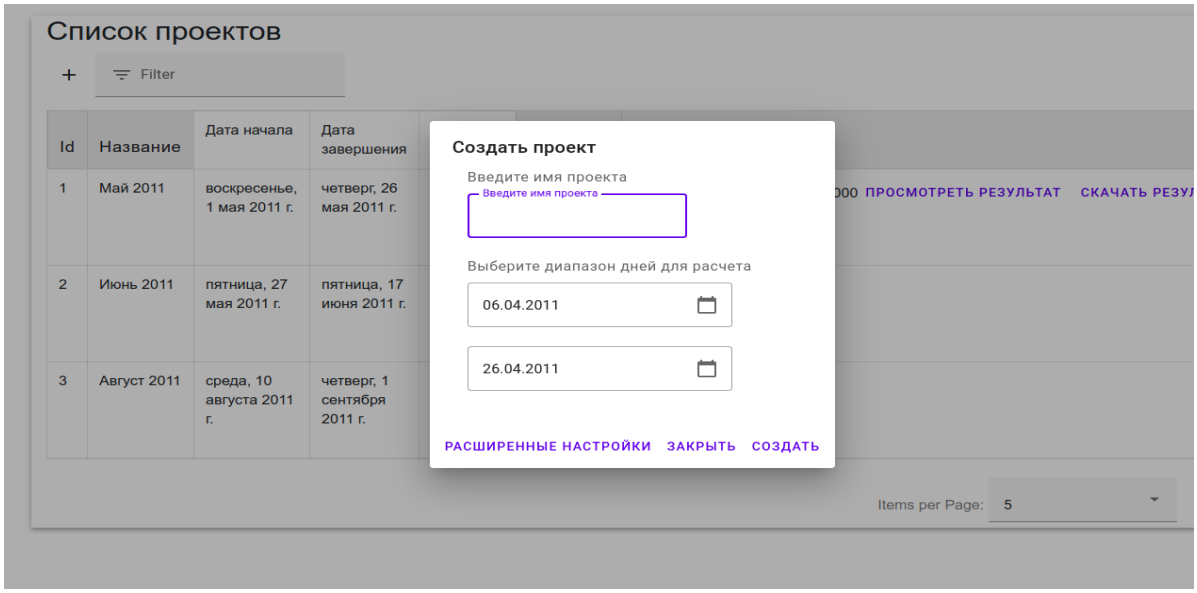

Рисунок 59 – Создание проекта

После создания проекта, в фоне уже начинается процесс подгрузки ГНСС-данных и вспомогательных файлов необходимых для расчетов.

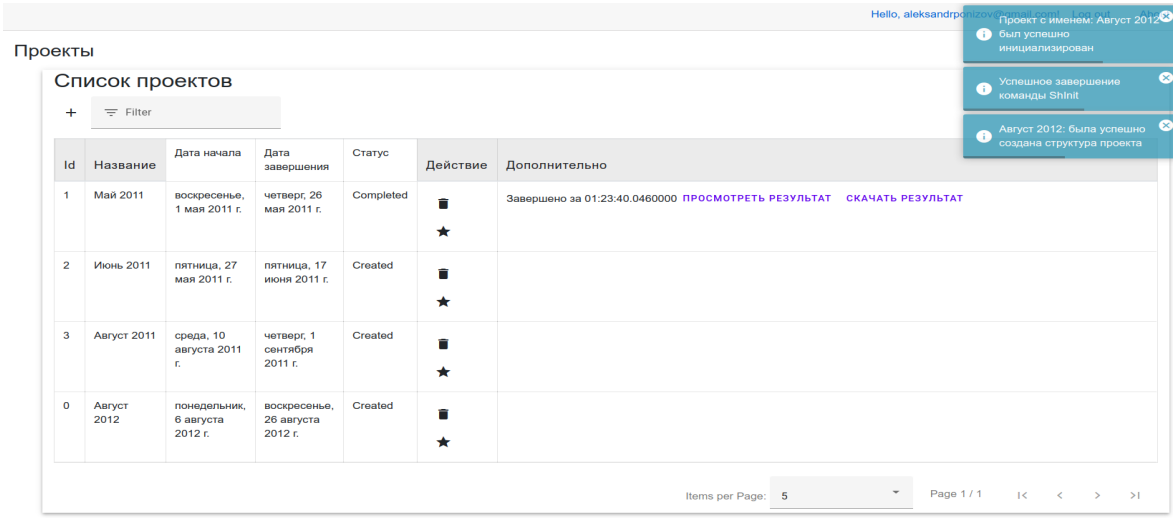

Рисунок 60 – Уведомление пользователя об успешно выполненных операциях

После запуска обработки проекта появляется соответствующий значок ожидания и статус проекта меняется на «Running», также начинается отсчет таймера, который показывает какое время проект находится в работе. По окончанию, можно либо загрузить результаты, либо посмотреть их в приложении (рисунок 61).

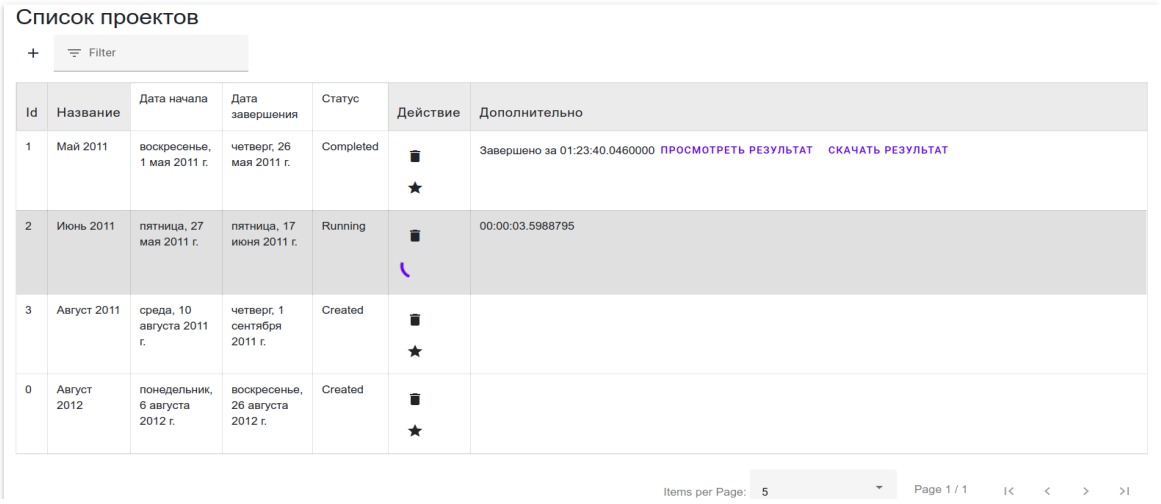

Рисунок 61 – Запущенная обработка проекта

Результаты, выдаваемые приложением полностью идентичны результатам выдаваемым пакетом GamitGlobk, так как внутри микросервис использует его для математической обработки.

ЗАКЛЮЧЕНИЕ

Современное состояние наук о земле обуславливает необходимость привлечения в качестве отправных данных при построениях результатов, полученных с помощью точных математических методов, выполненных на основе строгих методологических подходов. Эти данные должны удовлетворять критериям однозначности, обоснованности, верифицируемости и формальной непротиворечивости. Таким критериями, в случае изучения происходящих на земной поверхности смещений, полностью удовлетворяет метод космической геодезии с применением глобальных навигационных спутниковых систем. Однако, он является трудоемким и сложно-реализуем без помощи современного программно-аппаратного комплекса.

На первом этапе магистерской работы был произведён анализ предметной области, в итоге которого были рассмотрены: актуальность задачи обработки ГНСС-данных, применение глобальных навигационных спутниковых систем для измерения геодинамической активности, структура ГНСС-данных и произведен сравнительный анализ: существующих программных решений, микросервисной архитектуры.

На втором этапе (алгоритмическое и программное обеспечение решения задачи) был проведен анализ требований к микросервисам, определены основные функции требуемые к реализации в программном обеспечении, рассмотрено применение брокера сообщений для микросервисного взаимодействия. Произведено проектирование микросервисов, в рамках которого расмотрены варианты взаимодействия пользователя с приложением. Рассмотрен алгоритм обработки входных данных и обоснован выбор средств реализации приложения. Выполнена характеристика выбранного программно-технического обеспечения.

В рамках третьего этапа (реализация и тестирование приложения обработки ГНСС-данных) было произведена практическая разработка программного

продукта, согласно выбранной методологии разработки приложения, в рамках которой были разработаны следующие программные модули: модуль сбора ГНСС-данных, модуль конвертирования ГНСС-файлов в RINEX формат, модули приложения систематизации, сбора и хранения данных, модули приложения обработки ГНСС-данных, разработан графический интерфейс. Следующим этапом являлся этап компиляции и создания образов микросервисов которые совместно и реализуют функционал приложения обработки ГНСС-данных. Согласно методологии разработки было произведено тестирование и отладка программного продукта. Достоверность и практическая значимость полученного результата расмотрены в пунктах 3.4 и 3.5. В результате было получено приложение обработки ГНСС данных, состоящее из микросервисов и реализующее требуемый функционал.

Полученное приложение обработки ГНСС-данных, созданное на основе концепций микросервисной архитектуры и применении проверенного временем математического пакета GamitGlobk, значительно упростило процессы хранения, конвертирования и обработки таких данных. Уменьшило общее время на произведение расчетов с недели до нескольких часов. Полученный результат удовлетворяет поставленным целям и задачам работы.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

2 Жижерин В.С., Серов М.А., Холобуда С.П. Моделирование геодинамических процессов Верхнего Приамурья на основе GPS данных // Успехи современного естествознания. 2018. № 11. С. 103-108.

3 Там же, С.9

4 Начало формирования единой сети геодинамических наблюдений. / Быков В. Г., Бормотов В.А., Коковкин А.А [и др.]. – Вестник ДВО РАН, 2009, № 4. С. 83-93.

5 Инструментальное и информационно-технологическое обеспечение сейсмологических наблюдений на Дальнем Востоке России / Ханчук А.И., Коновалов А. В., Сорокин А. А. [и др.]. – Вестник ДВО РАН, №3, 2011. С.127-137.

6 Там же, С.11

7 Кинематика амурской литосферной плиты по данным GPS-геодезии. / Ашурков С.В., Саньков В.А., Мирошниченко А.И [и др.]. Геология и геофизика, 2011, №2. – 299-311 с.

8 Начало формирования единой сети геодинамических наблюдений. / Быков В. Г., Бормотов В.А., Коковкин А.А [и др.]. – Вестник ДВО РАН, 2009, № 4. С. 83-93.

9 Жижерин В.С., Серов М.А., Холобуда С.П. Моделирование геодинамических процессов Верхнего Приамурья на основе GPS данных // Успехи современного естествознания. 2018. № 11. С. 103-108.

10 Там же, С.17

11 Начало формирования единой сети геодинамических наблюдений. / Быков В. Г., Бормотов В.А., Коковкин А.А [и др.]. – Вестник ДВО РАН, 2009, № 4. С. 83-93.

12 Там же, С.18

13 Инструментальное и информационно-технологическое обеспечение сейсмологических наблюдений на Дальнем Востоке России / Ханчук А.И., Конова-

лов А. В., Сорокин А. А. [и др.]. – Вестник ДВО РАН, №3, 2011. С.127-137.

14 Архитектура микросервисов [Электронный ресурс]. URL: https://habr.com/ru/company/mailru/blog/320962/

15 CQRS. Факты и заблуждения [Электронный ресурс]. URL: https://habr.-com/ru/post/347908/

16 Gamit/Globk Documentation. [Электронный ресурс]. URL: http://geowe-b.mit.edu/gg/docs.php

17 Чистый код: создание, анализ и рефакторинг. Библиотека программиста. / Пер. с англ. – СПб.: Питер, 2010. – 464 с.

18 Персиваль Г. Python. Разработка на основе тестирования. / пер. с англ. Логунов А. В. – М.: ДМК Пресс, 2018. – 622 с.:

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Архитектура микросервисов [Электронный ресурс]. URL: https://habr.com/ru/company/mailru/blog/320962/

2 Босуэлл Д., Читаемый код, или Программирование как искусство. / Босуэлл Д., Фаучер Т – СПб.: Питер, 2012. – 208 с.: ил.

3 Влацкая И.В. Проектирование и реализация прикладного программного обеспечения [Электронный ресурс]: учебное пособие/ И.В. Влацкая, Н.А. Заельская, Н.С. Надточий – Электрон. текстовые данные. – Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2015. – 119 с. – Режим доступа: http://www.iprbookshop.ru/54145.html. – ЭБС «IPRbooks»

4 Жижерин В.С., Моделирование геодинамических процессов Верхнего Приамурья на основе GPS данных // Успехи современного естествознания. /Жижерин В.С., Серов М.А., Холобуда С.П. – 2018. № 11. С. 103-108.

5 Инструментальное и информационно-технологическое обеспечение сейсмологических наблюдений на Дальнем Востоке России / Ханчук А.И., Коновалов А. В., Сорокин А. А. [и др.]. – Вестник ДВО РАН, №3, 2011. С.127-137.

6 Информационные системы: Учебное пособие. / Е. В. Бурцева [и др.]. – Тамбов: ТГТУ, 2009. – 128 с.

7 Кинематика амурской литосферной плиты по данным GPS-геодезии. / Ашурков С.В., Саньков В.А., Мирошниченко А.И [и др.]. Геология и геофизика, 2011, №2. – 299-311 с.

8 Клири Стивен Конкурентность в C#. Асинхронное, параллельное и многопоточное программирование. 2-е межд. изд. – СПб.: Питер, 2020. – 272 с.

9 Маккарти, Джим. Правила разработки программного обеспечения [Текст] : практ. рук. : пер. с англ. / Д. Маккарти, М. Маккарти. - М.: Рус. Редакция; СПб. : Питер, 2007. - 220 с. + 1 эл. опт. диск (CD-ROM). - Предм. указ.: с. 216.

10 Начало формирования единой сети геодинамических наблюдений. / Быков В. Г., Бормотов В.А., Коковкин А.А [и др.]. – Вестник ДВО РАН, 2009, № 4.

С. 83-93.

11 Объектно-ориентированное программирование: учебное пособие для прикладного бакалавриата / А. Ф. Тузовский. – Москва: Издательство Юрайт, 2019. – 206 с. – (Университеты России). – ISBN 978-5-534-00849-4. – Текст: электронный // ЭБС Юрайт [сайт]. – URL: https://biblio-online.ru/bcode/434045 (дата обращения: 31.05.2019).

12 Персиваль Г. Python. Разработка на основе тестирования. / пер. с англ. Логунов А. В. – М.: ДМК Пресс, 2018. – 622 с.: ил.

13 Понизов А.В., Серов М.А., Галаган Т.А. Разработка Веб-приложения для обработки ГНСС-данных с использованием микросервисной архитектуры // Вестник АмГУ 2020. № 89. С. 27-31.

14 Понизов А.В., Галаган Т.А. Проектирование веб-приложения для обработки ГНСС-данных с использованием микросервисной архитектуры. //. Материалы 21 региональной научно-практической конференции «Молодежь 21 века: Шаг в бущее», Благовещенск, 2020, №4, С. 132-133.

15 Понизов А.В., Галаган Т.А. Преимущества архитектурного CQRS-подхода в высоконагруженных приложениях. // Материалы 14 международной научной конференции «САМ 2020», Благовещенск, 2020, С.81-84.

16 Просто о микросервисах. Блог компании Райфайзен банк. [Электронный ресурс]. URL: https://habr.com/ru/company/raiffeisenbank/blog/346380/

17 Принципы, паттерны и методики гибкой разработки на языке C#. / Пер. с англ. – СПб.: Символ-Плюс, 2011. – 768 с.

18 Ричардсон, Крис. Микросервисы. Паттерны разработки и рефакторинга / Ричардсон Крис – СПб.: Питер, 2019. – 544 с.

19 Сильнейшее современное землетрясение в Верхнем Приамурье 14 октября 2011 г.: первые результаты исследования. / Ханчук А.И., Сафонов Д.А., Коновалов А.В [и др.]. – Доклады академии наук, 2012, Т. 445, № 3. – 338–341 с.

20 Синицын С.В. Верификация программного обеспечения [Электронный ресурс]: чебное пособие / С.В. Синицын, Н.Ю. Налютин. – Электрон. текстовые данные. – Москва, Саратов: Интернет-Университет Информационных Техноло-

гий (ИНТУИТ), Вузовское образование, 2017 – 368 c. – 978-5-4487-0074-3. – Режим доступа: http://www.iprbookshop.ru/67396.html

21 Тепляков С. Паттерны проектирования на платформе .NET. – СПб.: Питер, 2015 – 320с.: ил. ISBN 978-5-496-01649-0

22 Технология разработки программного обеспечения [Электронный ресурс]: сб. учеб.-метод. материалов для направления подготовки 09.04.04 "Программная инженерия" / АмГУ, ФМиИ; сост. Т. А. Галаган. - Благовещенск: Изд-во Амур. гос. ун-та, 2017. - 51 с. http://irbis.amursu.ru/DigitalLibrary/AmurSU_Edition/10382.pdf

23 Технология разработки программного обеспечения [Электронный ресурс]: учеб. – метод. пособие / АмГУ, ФМиИ ; сост. Т. А. Галаган. – Благовещенск: Изд-во Амур. гос. ун-та, 2015. – 49 с. http://irbis.amursu.ru/DigitalLibrary/AmurSU_Edition/6799.pdf

24 Чамберс, Д. ASP.NET Core. Разработка приложений / Чамберс Д., Пэкетт Д., Тиммс С. – СПб.: Питер, 2018. – 464 с.

25 Чистый код: создание, анализ и рефакторинг. Библиотека программиста. / Пер. с англ. – СПб.: Питер, 2010. – 464 с.

26 Analysis of the far-feld crustal displacements caused by the 2011 Great Tohoku earthquake inferred from continuous GPS observations / Shestakov N.V., Takahashi H., Ohzono M., Prytkov A.S., Vasilenko N.F., Bykov V.G., Luneva M.N., Bormotov V.A., Gerasimenko M.D., Gerasimov G.N., Kolomiets A.G., Baek J., Park P.-H., Serov M.A. – Tectonophysics. 2012, T. 524-525. - 76-86 c.

27 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. – СПб.: Питер, 2013. – 896 с.

28 CodinGame: Creating Web API in ASP .NET Core 2.0 [Электронный ресурс]. URL: https://www.codingame.com/playgrounds/35462/creating-web-api-in-asp-net-core-2-0/part-1---web-api

29 CQRS. Факты и заблуждения [Электронный ресурс]. URL: https://habr.-com/ru/post/347908/

30 DigitalOcean: Initial Server Setup with Ubuntu 18.04 [Электронный

ресурс]. URL: https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04

31 Gamit/Globk Documentation. [Электронный ресурс]. URL: http://geoweb.mit.edu/gg/docs.php

32 Linux Container – Proxmox VE. [Электронный ресурс].URL: https://pve.proxmox.com/wiki/Linux_Container

33 Medium.com: Setup Entity Framework Core [Электронный ресурс]. URL: https://medium.com/@balramchavan/setup-entity-framework-core-for-mysql-in-asp-net-core-2-5b40a5a3af94

34 Metanit.com: Паттерны проектирования в C# и .NET [Электронный ресурс]. URL: https://metanit.com/sharp/patterns/

35 Metanit.com: Полное руководство по языку программирования C# и платформе .NET 4.7 [Электронный ресурс]. URL: https://metanit.com/sharp/tutorial

36 Microsoft Docs: Документация к ASP .Net Core [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

37 Microsoft Docs: Документация к System.Net.Http [Электронный ресурс]. URL: https://docs.microsoft.com/enus/dotnet/api/syытem.net.http.httpclent?view =net framework-4.7.2

38 Official Knowledge Base by phoenixNAP: How To Configure MySQL 8.0 On Ubuntu 18.04 [Электронный ресурс]. URL: https://phoenixnap.com/kb/how-to-install-mysql-on-ubuntu-18-04

39 Telerik.com: Build and Deploy Your ASP.NET Core Application with Apache [Электронный ресурс]. URL: https://www.telerik.com/blogs/build-deploy-asp-net-core-application-apache