УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Информационная безопасность»

**Трек 1**

Проверка проекта на соответствие требованиям безопасности

*Требования к приложению HappyBirthdayBot*

Студент

*Митрофанов Е. Ю.*

*P34101*

Преподаватель

*Маркина Т. А.*

Санкт-Петербург, 2022 г.

Оглавление

[Общие сведения о приложении 2](#_Toc122026181)

[UseCase диаграмма 3](#_Toc122026182)

[Стек разработки 3](#_Toc122026183)

[Компоненты приложения 4](#_Toc122026184)

[Архитектура приложения 4](#_Toc122026185)

[Диаграмма структуры базы данных 5](#_Toc122026186)

[Пример обработки запроса 6](#_Toc122026187)

[Общая архитектура приложения 7](#_Toc122026188)

[UML – диаграмма классов 7](#_Toc122026189)

[Требования безопасности 8](#_Toc122026190)

[Требования к тестированию приложения 9](#_Toc122026191)

[Список функциональных требований с точки зрения системы 10](#_Toc122026192)

[Описание разработанных тестовых сценариев 11](#_Toc122026193)

[Степень покрытия тестами 13](#_Toc122026194)

[Внесенные исправления 13](#_Toc122026195)

# Общие сведения о приложении

Телеграмм – бот @ITMOHappyBirthdayBot разработан с целью введения в мессенджер функции напоминания о ближайших Днях Рождения друзей, коллег и собеседников из групповых чатов.

На данный момент бот поддерживает следующие функции:

* Автоматическое добавление пользователя в базу данных
* Заполнение дополнительной информации о пользователе:
  + Дата рождения
  + Опциональный редактируемый список желаний
* Добавление бота в чат и связывание пользователей из чата друг с другом
* Добавление пользователя в друзья без привязки к чату
* Автоматические напоминания о предстоящих Днях Рождения в личных сообщениях с предоставлением списка желаний
* Настройка получения уведомлений позволяющая выбрать период времени, определяющий насколько заранее получать уведомления
* Возможность просмотреть все Дни Рождения в ближайший месяц

Бот с инструкцией доступен в Телеграмме по тегу [@ITMOHappyBirthdayBot](https://t.me/ITMOHappyBirthdayBot)

Приложение и все его компоненты развернуто на облачном сервисе Yandex Cloud

## UseCase диаграмма

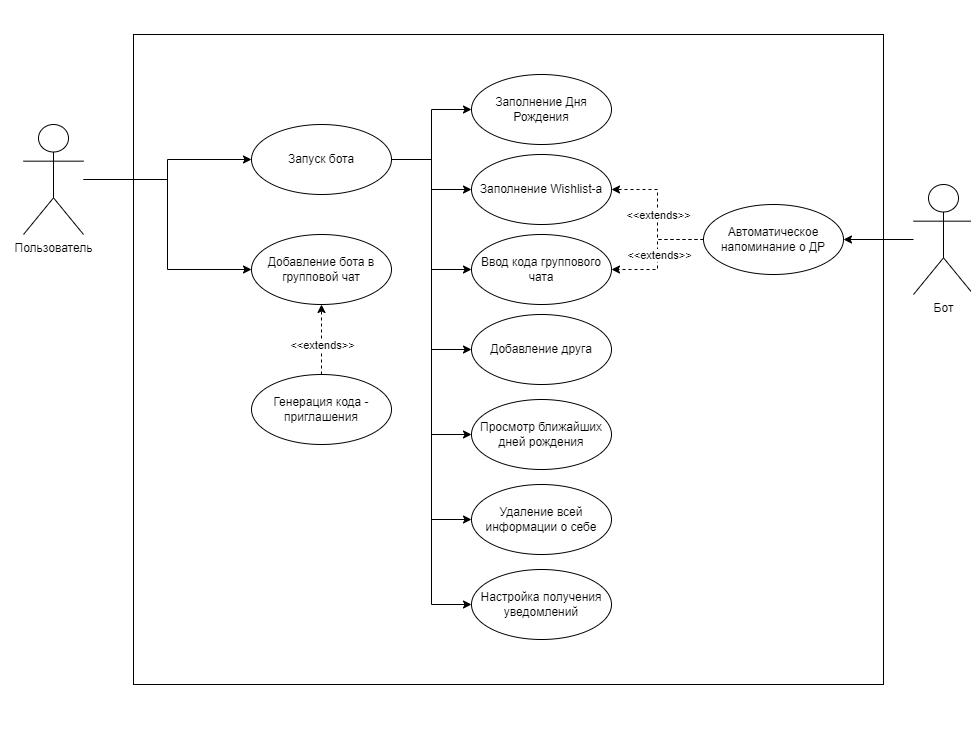


Рисунок 1, UseCase диаграмма приложения

# Стек разработки

Бот разработан на языке Java и использует следующий стек технологий:

* Spring Boot – фреймворк для разработки и запуска stand-alone Spring приложений.
* Telegram Bot Java Library – библиотека для создания Телеграмм ботов на языке Java
* PostgreSQL – open-source база данных, обеспечивающая весь необходимый функционал для персистентного хранения информации
* Hibernate – библиотека для задач объектно-реляционного отображения и работы с базами данных
* RabbitMQ – программный брокер сообщений, используется в проекте для асинхронной обработки рассылок

# 

# Компоненты приложения

Приложение использует Spring Boot приложение для обработки входящих обновлений бота. Подробнее про архитектуру приложения в следующем пункте

Для персистентного хранения данных используется база данных PostgreSQL, управляемая системой миграций FlyWay

Асинхронный обмен сообщениями и обработки обновлений, приходящих из Telegram, используется брокер сообщений RabbitMQ с настроенными очередями

Все компоненты приложения собраны в Docker – контейнер и развернуты на виртуальной машине под управлением Ubuntu 18

# Архитектура приложения

Архитектура приложения представляет собой монолит с применением MVC структуры для ослабления связей и повышения готовности к масштабированию.

В соответствии с MVC архитектурой приложение разделяется на три уровня:

1. Обмен данными происходит через спецификацию, предоставляемую Телеграмм, а также через библиотеку Telegram Bot Java Library. Передаваемые объекты – DTO классы со всей информацией о принятом сообщении или действии с ботом.
2. Бизнес-логика в зависимости от типа действия или команды обрабатывает запрос и при необходимости обращается через сервисный уровень к информационной системе (базе данных).
3. В зависимости от команды или данных пользователь через Telegram API получает необходимое сообщение или отчет о выполненном действии.

Сервисный уровень реализует работу с данными и их хранение в базе данных через репозитории.

Отдельно стоит отметить асинхронную отправку и получение сообщений через брокера. Такой подход позволит не потерять данные и обеспечит повышенную производительность.

Выбор монолитной архитектуры обоснован бизнес-логикой приложения. Каждое действие пользователя является неразрывной логической единицей и обрабатывается одним набором контроллеров и одной базой данных. Асинхронность для обмена сообщениями реализована через программный брокер сообщений.

## Диаграмма структуры базы данных

На рисунке приведена схема таблиц разработанного приложения [Рисунок 2]

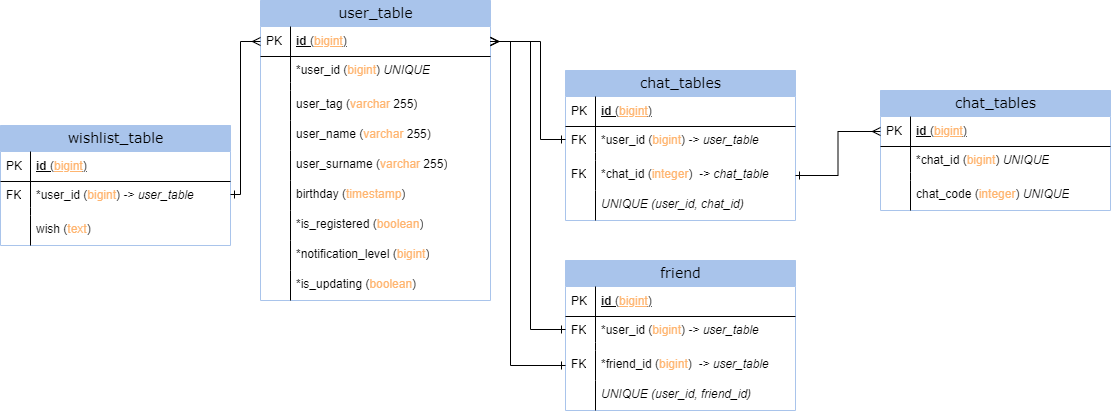


Рисунок 2, схема базы данных

## Пример обработки запроса

Рассмотрим для примера запуск бота с помощью команды */start* [Рисунок 3]

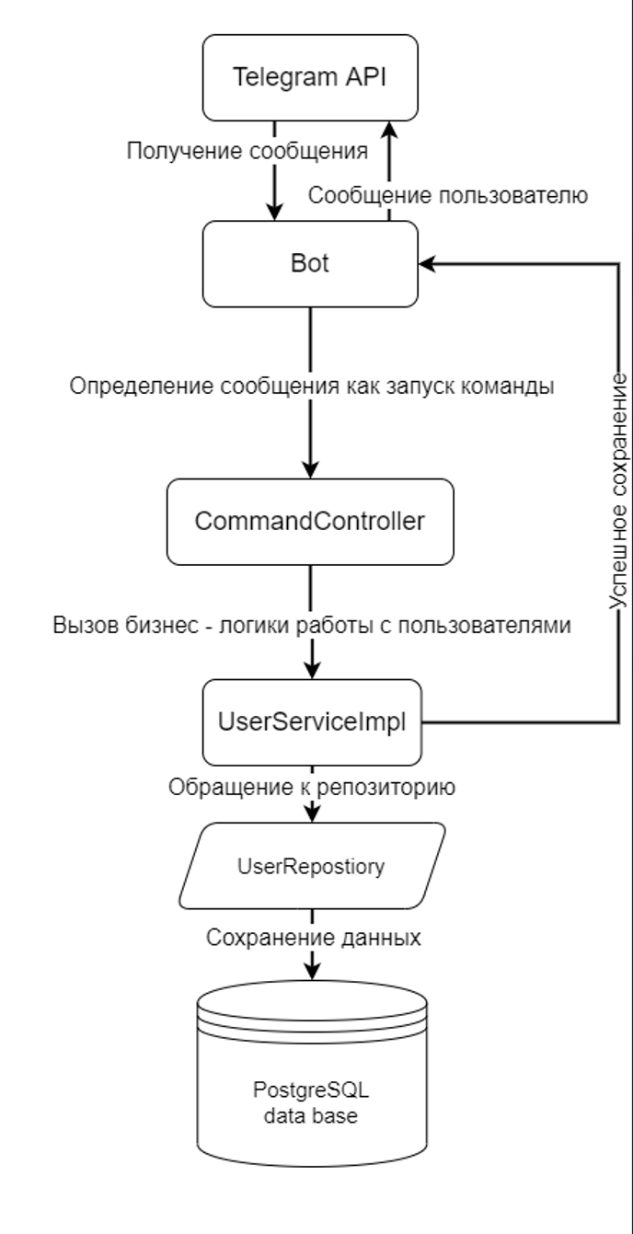
1. Класс Bot, обрабатывающий обновления и действия, совершенные с ботом, получает новое сообщение, включающее текст и метаинформацию о пользователе, отправившим его, и другую информацию, к примеру время отправки.

Рисунок 3, диаграмма обработки команды /start

1. Парсинг текстового сообщения позволяет понять, что вызвана команда, после чего вызывается контроллер команд с запуском конкретного обработчика.
2. Контроллер вызывает определенную бизнес – логику. В случае *команды /start* это добавление пользователя в базу данных (если его там еще нет) и отправка приветственного сообщения.
3. После чего соответствующий repository класс сохраняет полученную метаинформацию в базу данных.

## Общая архитектура приложения

На рисунке представлена монолитная архитектура приложения [Рисунок 2]

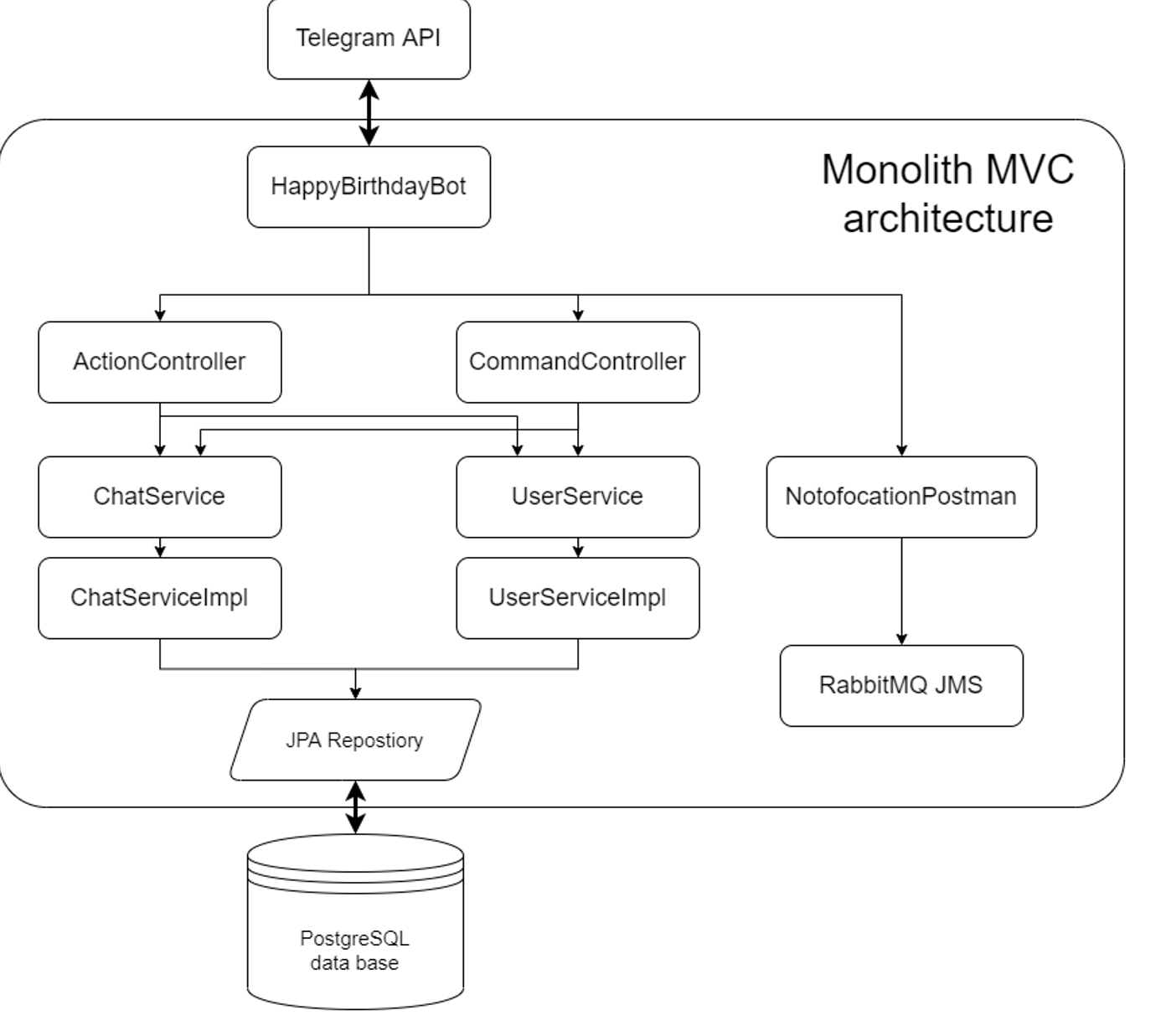


Рисунок 4, архитектура приложения

## UML – диаграмма классов

На рисунке представлена подробная диаграмма классов [Рисунок 3]

Изображение выглядит как текст, монитор, черный, экран

Автоматически созданное описание

Рисунок 5, UML диаграмма классов

# Требования безопасности

Для разрабатываемого приложения можно выделить следующие требования:

1. Код должен не иметь уязвимостей, описанных в [Common Weakness Enumeration](https://cwe.mitre.org/)
2. Все версии зависимостей должны иметь Long Time Support версию
3. Код приложения должен быть подвержен обфускации – приведен к виду, затрудняющему анализ и понимание, но сохраняющему всю функциональность
4. Исходный код не должен содержать конфиденциальную информацию: Token телеграмм – бота, данные для подключения к базе данных и т. д.
5. Telegram Bot должен соответствовать требованиям безопасности для разработчиков - [Security Guidelines for Developers](https://core.telegram.org/mtproto/security_guidelines) В большей степени необходимо обратить внимание на обработку и дешифровку сообщений – разделы «Checking message length», «Checking session\_id» и «Checking msg\_id». Выпросы шифрования и обмена ключами неактуальны, так как при разработке используется официальный клиент и набор готовых инструментов для разработки бота.
6. Приложение должно быть протестировано разными способами: статический анализ кода и автоматизированные модульные и интеграционные тесты (подробнее в следующем разделе)
7. Приложение должно иметь доступ только для авторизованных пользователей

Стоит отметить, что последний пункт полностью обеспечивается использованием Telegram API и той степенью защиты, что он предоставляет. Но параллельно необходимо придерживаться гайдлайнов, указанных в пункте [5.]

# Требования к тестированию приложения

Разработанное приложение должно быть протестировано следующими способами:

1. Статический анализ кода

*Подобный анализ позволит избежать RunTime ошибок, связанных с переполнением, обращению к null – объектам, утечки памяти, ошибок форматирования и ошибок компиляции*

*Для выполнения подойдут такие внешние инструменты как ReSharper, а также статические анализаторы встроенные в среду разработки, к примеру в IntelliJ IDEA.*

1. Инспекция кода

*На этом этапе мы должны обратить внимание на поиск проблемных мест и анализа поведения приложения в конкретных ситуациях. Необходимо проследить, чтобы приложение не содержало невыполняемых участков кода, лишних методов и бизнес – логики и не имело логических ошибок, после чего провести рефакторинг кода в этих местах.*

1. Функциональное тестирование, включающее:
   1. Модульное тестирование приложения

*На этом этапе необходимо протестировать независимые модули, к примеру асинхронную отправку сообщений, работу с базой данных и другие меньшие модули, к примеру парсинг сообщений и команд. Для Java приложений стандартом является использование библиотеки JUnit*

* 1. Интеграционное тестирование приложения

*На этом этапе происходит проверка работоспособности всего приложения. Необходимо проверить совместную работу разных уровней приложения, работу бизнес-логики и интеграцию с внешним API. Необходимо составить тестовые кейсы каждому из которых соотнести ожидаемый результат.*

*Тестирование проводится в том числе с помощью Java библиотеки Mockito, позволяющей создавать «заглушки» на некоторые модули и проверять именно интеграционное взаимодействие*

* 1. Фаззинг – тестирование приложения

*В данном случае тестирование Telegram – бота сводится к отправке неожиданных и неправильных данных. Нестандартные символы, невалидные данные (дата, имя), попытки сломать бота, отправив некорректный файл или совершить ряд нестандартных операций. Бот должен обрабатывать любой набор данных, в противном случае уведомляя пользователя о «плохом запросе»*

Тестирование заключается в проверке именно разработанного приложения. Функции, предоставляемые внешним сервисом и библиотеками, в нашем случае Telegram API и библиотека Telegram Bots по заверению разработчиков соответствуют стандартам безопасности и содержат встроенные инструменты автоматического тестирования и не нуждаются в повторном тестировании.

# Список функциональных требований с точки зрения системы

1. Система должна предоставлять пользователю возможность запуска и последующей регистрации.

*Информация о пользователе автоматически берется из API Telegram, дополнительные данные вводятся самостоятельно. Для хранения используется локальная база данных*

1. Система должна предоставлять возможность объединения пользователей по групповым чатам, с дальнейшей автоматической отправкой напоминаний для всех участников.

*Для хранения используется локальная база данных, для получения информации о групповом чате используется API Telegram.*

1. Система должна предоставлять возможность настройки получаемых уведомлений и задания периода получения.

*Для хранения используется локальная база данных.*

1. Система должна предоставлять возможность создания персонального wishlist-а для предоставления рекомендаций*.*

*Для хранения используется локальная база данных.*

1. Система должна предоставлять возможность просмотра ближайших напоминаний в следующие 30 дней.

*Для получения информации используется локальная база данных.*

1. Система должна в автоматическом режиме рассылать напоминания в зависимости от выставленных пользователем настроек.

*Для получения информации используется локальная база данных, отправка напоминаний происходит через API Telegram.*

1. Система должна предоставлять возможность добавления пользователя в список друзей, для получения напоминаний без привязки к групповому чату.

*Для получения информации используется локальная база данных, отправка напоминаний происходит через API Telegram.*

# Описание разработанных тестовых сценариев

Для приложения согласно всем функциональным требованиям [Стр. 10] и требованиям к тестированию [Стр. 9] разработаны автоматизированные тесты, использующие независимую локальную базу данных и тестовые данные несуществующих пользователей.

Разработаны следующие тесты:

1. Интеграционные тесты, проверяющих общее взаимодействие всех функций и компонентов приложения, включая некоторые открытые API функции Telegram, обмен сообщений через Message Broker и также согласованное хранение всех данных в локальной базе данных.
2. Модульные тесты, проверяющие работу отдельных логических функций, отвечающий за парсинг и формат дат, обработку запросов и команд, валидацию входящих и исходящих сообщений, обработку исключений и ошибок.
3. Модульные и интеграционные тесты, проверяющие обработку приложением неверных данных, пытающихся нарушить работу. К примеру, несуществующие комбинации имен пользователей, дней и месяцев, невалидные ответные запросы - нажатие кнопок, прикрепленных к сообщениям, блокировка бота, удаление персональных данных, изменение существующих данных и т. д.

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунок 6, демонстрация прохождения всех тестовых сценариев

## Степень покрытия тестами

В процессе разработки тестовых сценариев удалось достичь тестового покрытия в 93%. Степень покрытия методов приложения меньше, так как приложение использует достаточное количество сторонних готовых методов и функций из библиотек Telegram, проверка работоспособности которых не является необходимой при разработке бота и, кроме того, невозможна без создания собственного Telegram – клиента.

При разработке интеграционных тестов использовалась технология Mockito для эмуляции работы этих методов.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7, автоматический прогон тестовых сценариев при сборке приложения

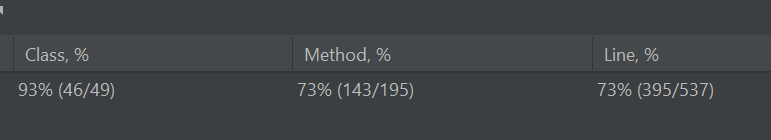


Рисунок 8, демонстрация тестового покрытия

## Внесенные исправления

Разработка тестовых сценариев позволила обнаружить незащищенные места в приложении. Это касается, к примеру, использования нестандартных символов, невиладных форматов сообщений, обработку некоторых ошибок, возникающих при несогласованности данных, обновление информации в локальной базе данных и некоторые некорректные запросы для получения информации из нее.

В процессе работы выявленные неполадки и «слабые» места были исправлены или переписаны.