УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Рефакторинг баз данных и приложений»

**Проект**

# Разработка Telegram бота "HappyBirthdayBot"

Студент

*Митрофанов Е. Ю.*

*P34101*

Преподаватель

*Логинов И. П.*

Санкт-Петербург, 2022 г.

# Оглавление



# 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

Приложение и все его компоненты развернуто на облачном сервисе Yandex Cloud

# 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.

## *Пример обработки запроса*

Рассмотрим для примера запуск бота с помощью команды */start* [Рисунок 1]

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

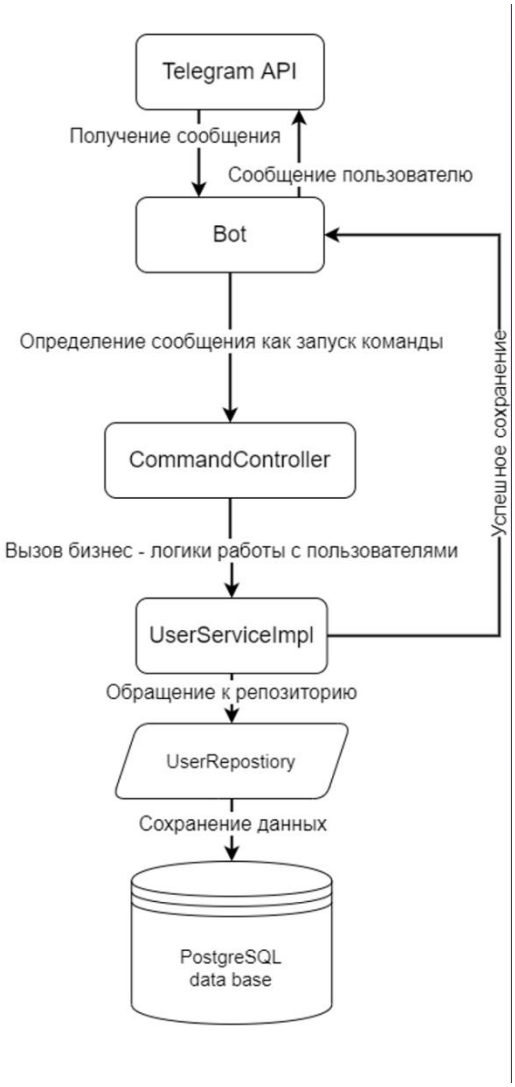


*Рисунок 1, диаграмма обработки команды /start*

## *Общая архитектура приложения*

На рисунке представлена монолитная архитектура приложения [Рисунок 2]

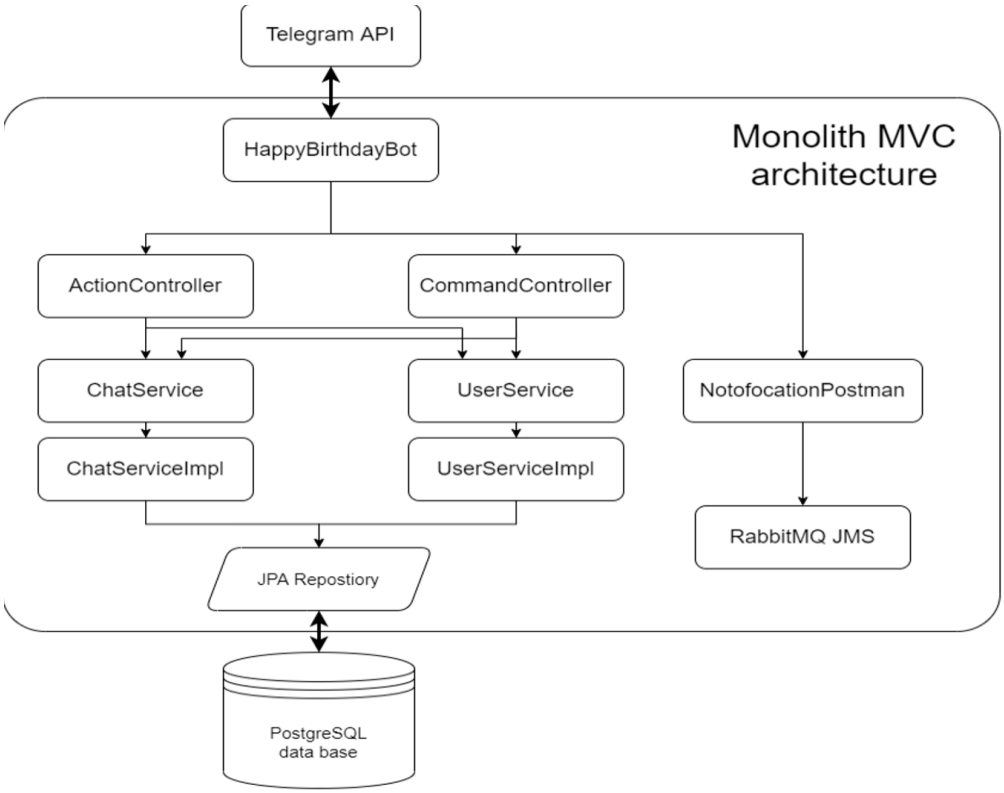


*Рисунок 2, архитектура приложения*

## *UML – диаграмма классов*

На рисунке представлена подробная диаграмма классов [Рисунок 3]

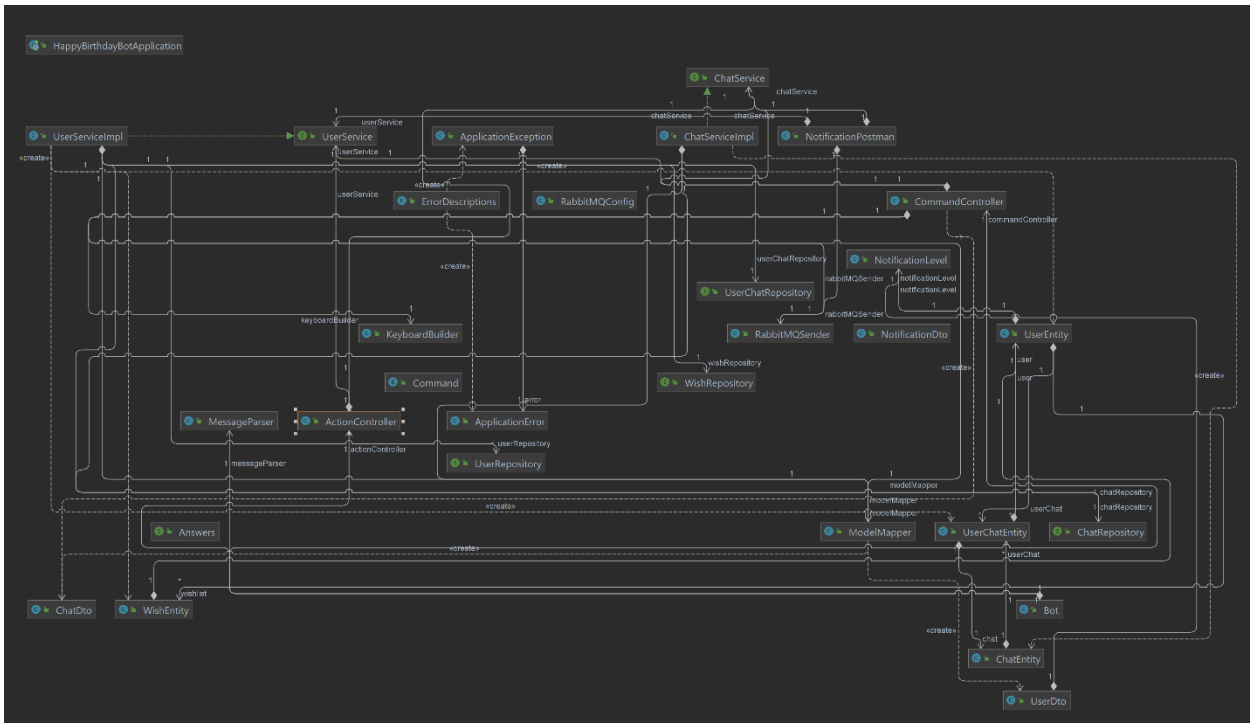


*Рисунок 3, UML диаграмма классов*

# Требования безопасности

Для разрабатываемого приложения можно выделить следующие требования:

1. Код должен не иметь уязвимостей, описанных в Common Weakness Enumeration
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
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.]

# Тестирование 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.*

2. 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.*

3. Функциональное тестирование, включающее:

   a. 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*

   b. 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*

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.