



GeekBrains

Многофункциональный Telegram-бот

«Dogobot – удаленный помощник»

Программа:	Разработчик
Специализация:	Программист
ФИО: Балинян Давид Ваграмович	

Москва

2024 год

Содержание

Содержание.....	2
Введение.....	5
Глава 1. Применяемые технологии и методики.....	7
Глава 2. Задачи, описание и возможности чат-бота	12
Глава 3. Структура программы.....	18
3.1 Общий состав проекта	18
3.2 Алгоритм взаимодействия с пользователем в общих чертах	21
3.3 Файл pom.xml и необходимые зависимости	22
3.4 Содержимое папки resources.....	25
3.4.1 Файл конфигурации application.properties.	26
3.4.2 Файл конфигурации логов logback.xml	29
3.4.3 Эмблема приложения.....	30
Глава 4 Настройка, сборка и запуск проекта.....	31
4.1 Файл – application.properties.....	31
4.1.1 Все параметры, которые необходимо настроить	32
4.1.2 Параметры telegrambot.	33
4.1.3 Параметры spring.datasource и spring.jpa	34
4.1.4 Параметры email.....	35
4.2 Файл JSON с пользовательскими настройками	36
4.3 Пути к логам и скриншотам.....	37
4.4 Сборка проекта и его запуск.	38
4.4.1 Сборка проекта	38
4.4.2 Запуск проекта.....	39
Глава 5 Подробнее о классах	41

5.1 Класс TelegramBot.....	41
5.1.1 Общая информация.....	41
5.1.2 «Траектория движения запроса»	43
5.2 Классы FileManager, FileDir и др.	45
5.3 Классы пакета config	48
Заключение	49
Список используемой литературы	50
Приложения	51
Приложение 01. Содержимое класса DogobotApplication	51
Приложение 02. Пакет config, класс BotConfig	52
Приложение 03. Пакет config, класс BotInitializer	53
Приложение 04. Пакет config, класс EmailConfig.....	54
Приложение 05. Пакет config, класс UserConfig.....	55
Приложение 06. Пакет exception, класс EmailerException	60
Приложение 07. Пакет exception, класс FilerException	61
Приложение 08. Пакет model, класс FileDir	62
Приложение 09. Пакет model, класс User	68
Приложение 10. Пакет model, класс UserRepository	71
Приложение 11. Пакет service, класс Archiver	72
Приложение 12. Пакет service, класс Emailer.....	77
Приложение 13. Пакет service, класс FileManager.....	83
Приложение 14. Пакет service, класс Filer.....	112
Приложение 15. Пакет service, класс Jsoner	116
Приложение 16. Пакет service, класс Screenshoter.....	119
Приложение 17. Пакет service, класс Terminaler	122

Приложение 18. Пакет service, класс Userer.....	125
Приложение 19. Пакет view_controller, класс TelegramBot.....	129
Приложение 20. Файл pom.xml.....	161
Приложение 21. Папка resources, файл logback.xml.....	165
Приложение 22. Папка resources, файл application.properties	167

Введение

Проект представляет из себя описание и разработку Java-приложения – Telegram Bot «Dogobot – удалённый помощник» (далее **чат-бот**), позволяющее пользователю получить удаленный доступ к файловой системе на удалённом устройстве, а также обладающее возможностями выполнять действия на нём. На этапе написания проекта, **основной функционал чат-бота** предусматривает:

- основные функции файлового менеджера на удаленном устройстве:
 - навигация по файловой системе,
 - получение информации о файле или папке (далее ф.п.),
 - переименование ф.п.,
 - перемещение ф.п.,
 - копирование ф.п.,
 - удаление ф.п.;
- упаковка и распаковка ф.п., как с паролем, так и без него;
- получение любого файла прямо в чат-бот (соответствующий требованиям для передачи через Telegram);
- получение/отправка любого файла на электронную почту (соответствующий требованиям для передачи через электронную почту);
- создание и получение скриншота на удаленном устройстве;
- передачу команд для выполнения в терминале, как в ОС Windows (командная строка), так и в ОС Linux и получением ответа выполнения команды;
- удаленное редактирование личных настроек пользователя.

Пользователями чат-бота могут быть, как обычные пользователи персонального компьютера (далее ПК), желающие получить к нему удаленный доступ и иметь возможность произвести удаленные действия, так и системные администраторы, которые могут применять программу в своей работе.

Существует множество программ для удаленного доступа к ПК. В основном они удобные, обладают широким функционалом, они предоставляют

возможности, которые максимально приближены к тем же возможностям, что имеет пользователь, который физически находится рядом со своим ПК.

Но такие программы обладают определёнными свойствами и требованиями, которые являются недостатками при использовании удаленного доступа с мобильного устройства:

- для онлайн видео потока требуется хорошее и стабильное соединение с интернет;
- интерфейс популярных операционных систем спроектирован под эксплуатацию с большим экраном и наличием популярных устройств ввода: мышь и клавиатура. Экран мобильного устройства – маленький и чаще всего рядом нет ни клавиатуры, ни мыши, что приводит к неудобству эксплуатации;
- в большинстве случаев они платные или бесплатны, но с ограничениями.

Значит есть ряд сценариев эксплуатации программ для удаленного доступа, когда использование чат-бота, имеющего урезанный функционал от вышеупомянутых существующих программ, является **более целесообразным, быстрым, простым и бесплатным способом достижения цели** пользователя с мобильного устройства и не только с него.

Использованные инструменты на момент написания проекта: IntelliJ IDEA, Java Development Kit, Spring Initializr (SpringBoot), Apache Maven, DBeaver (PostgreSQL), Git и GitHub, Telegram.

Автор и программист проекта - Балинян Давид Ваграмович.

Ссылка на актуальный исходный код проекта в репозитории GitHub: <https://github.com/DavidZolotoi/Dogobot>.

Краткая видеопрезентация возможностей (рекомендуется установить максимальное качество воспроизведения, иначе шрифт может быть не читаем, а также там регулировка скорости): <https://cloud.mail.ru/public/xDYq/xWNNRu1w3>

Глава 1. Применяемые технологии и методики

Основным языком программирования в проекте выбран **Java**. Два определения для Java по состоянию на май 2024 года.

Определение с официального сайта <https://www.oracle.com/cis/java/>:

Java - язык программирования и платформа разработки № 1 в мире. Он уменьшает затраты, сокращает сроки разработки, стимулирует инновации и совершенствует сервисы приложений. Миллионы разработчиков, более 60 миллиарда работающих виртуальных машин Java во всем мире — Java продолжает оставаться предпочтительной платформой разработки для компаний и разработчиков.



Определение со свободной энциклопедии <https://ru.wikipedia.org/wiki/Java>:

Java - строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Для того, чтоб разрабатывать и запускать проекты, написанные на Java, требуется **Java Development Kit (сокр. JDK)** - бесплатно распространяемый корпорацией Oracle Corporation (ранее Sun Microsystems) комплект разработчика приложений на языке Java, включающий в себя **компилятор Java (javac)**, **стандартные библиотеки классов Java**, примеры, документацию, различные утилиты и **исполнительную систему Java (JRE)**. В состав JDK не входит интегрированная среда разработки на Java, поэтому разработчик, использующий только JDK, вынужден использовать внешний текстовый редактор и компилировать свои программы, используя утилиты командной строки. Скачать JDK можно с официального сайта <https://www.oracle.com/java/technologies/downloads/>.

Тут стоит упомянуть об **OpenJDK** - проект по созданию полностью совместимого Java Development Kit, состоящего исключительно из свободного и открытого исходного кода. Скачать можно с оф. сайта <https://jdk.java.net/>.

Разработка проекта ведется в ведущей IDE для разработки на языке Java - **IntelliJ IDEA** - интегрированной среде разработки программного обеспечения (для многих языков программирования, в частности Java, JavaScript, Python), разработанная компанией JetBrains. Скачать можно с оф. сайта <https://www.jetbrains.com/ru-ru/idea/>.



Для контроля версий выбраны **Git** с **GitHub** - крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Git - система контроля версий - это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.



Официальные сайты: <https://github.com/> и <https://git-scm.com/book/ru/v2>.

Использование системы контроля версий, в том числе позволило вести разработку на различных ПК в различных операционных системах, а также тестировать работу продукта в них: **OS Debian GNU/Linux 12 (bookworm)**, **OS Windows 10 Pro**, **OS Windows 11 Pro**.

Проект пишется с использованием популярного универсального фреймворка **Spring** с открытым исходным кодом для разработки приложений для Java-платформы (также существует форк для платформы .NET Framework, названный Spring.NET).



Он предоставляет обширный набор инструментов и функциональности для создания различных типов приложений, включая веб-приложения, микросервисы, приложения для обработки данных и многое другое. Spring обеспечивает инверсию управления (IoC), внедрение зависимостей (DI), а также поддерживает аспектно-ориентированное программирование (AOP). В рамках Spring Framework используется **Spring Boot**, который упрощает создание автономных, готовых к запуску приложений на основе Spring. Spring Boot предлагает конвенции по умолчанию, автоматическую конфигурацию и встроенные инструменты для управления зависимостями, что позволяет разработчикам быстро создавать приложения без необходимости тратить много времени на настройку. Spring Boot также облегчает развертывание приложений и управление внешними зависимостями.

Начальная структура проекта была создана с использованием инструмента **Spring Initializr** - <https://start.spring.io/> — это онлайн-инструмент, с помощью которого был создан новый проект на основе Spring Framework. Он предоставляет удобный интерфейс, где были выбраны необходимые зависимости (о них в следующих главах), настройки и версии, после чего была сгенерирована начальная структура проекта - архив, который был скачан и использован в дальнейшей разработке. Этот инструмент значительно упрощает начальную настройку проектов и ускоряет процесс разработки. Spring Initializr использует Maven в качестве инструмента для управления зависимостями и сборки проекта.

Apache Maven — это инструмент для управления проектами и сборки программного обеспечения на основе концепции управления зависимостями. Он позволяет автоматизировать процесс сборки



проекта, управлять зависимостями, выполнять тестирование и многое другое. Скачать можно с оф. сайта - <https://maven.apache.org/>.

Выбранные в Spring Initializr необходимые зависимости, автоматически добавлены в файл конфигурации Maven (pom.xml). Это позволяет при сборке проекта с помощью Maven, автоматически загрузить все необходимые зависимости из репозитория и включить в проект. Таким образом, Spring Initializr и Maven тесно интегрированы для упрощения создания и управления проектами на основе Spring Framework.

Тут также стоит упомянуть о концепции, к которой стремится разработка проекта - **Model-View-Controller (MVC, «Модель-Представление-Контроллер»)** - схема разделения данных приложения и управляющей логики на три отдельных компонента, модификация которых возможна независимо:

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

В качестве базы данных для использования в проекте выбрана – **PostgreSQL** — это мощная объектно-реляционная система управления базами данных (СУБД), которая широко используется для разработки приложений. Она предлагает множество продвинутых функций, таких как поддержка SQL, транзакций, индексирование, хранимые процедуры, репликация данных и многое другое. PostgreSQL является открытым исходным кодом и доступен бесплатно для использования. Благодаря своей надежности, производительности и расширяемости, PostgreSQL популярен среди разработчиков для создания надежных и масштабируемых баз данных. Скачать можно с официального сайта <https://www.postgresql.org/>. Для альтернативной работы с базой данных PostgreSQL (получение/редактирование



данных) вне разрабатываемого чат-бота были использованы инструменты DBeaver (чаще всего) и pgAdmin (редко).

DBeaver – это кроссплатформенный универсальный инструмент для работы с базами данных, который поддерживает множество СУБД, включая PostgreSQL. DBeaver предоставляет удобный графический интерфейс для выполнения SQL-запросов, просмотра и редактирования данных, управления структурой базы данных, импорта и экспорта данных и многое другое. Он предлагает широкий набор функций и инструментов для удобной работы с базами данных. Скачать можно с оф. сайта <https://dbeaver.io/>.



pgAdmin — это административный инструмент для управления базами данных PostgreSQL. Он предоставляет графический интерфейс для администрирования баз данных, включая создание и удаление баз данных, таблиц, индексов, пользователей, выполнение SQL-запросов, мониторинг производительности и многое другое. pgAdmin обеспечивает удобное и мощное средство для управления базами данных PostgreSQL. Скачать можно с оф. сайта PostgreSQL.

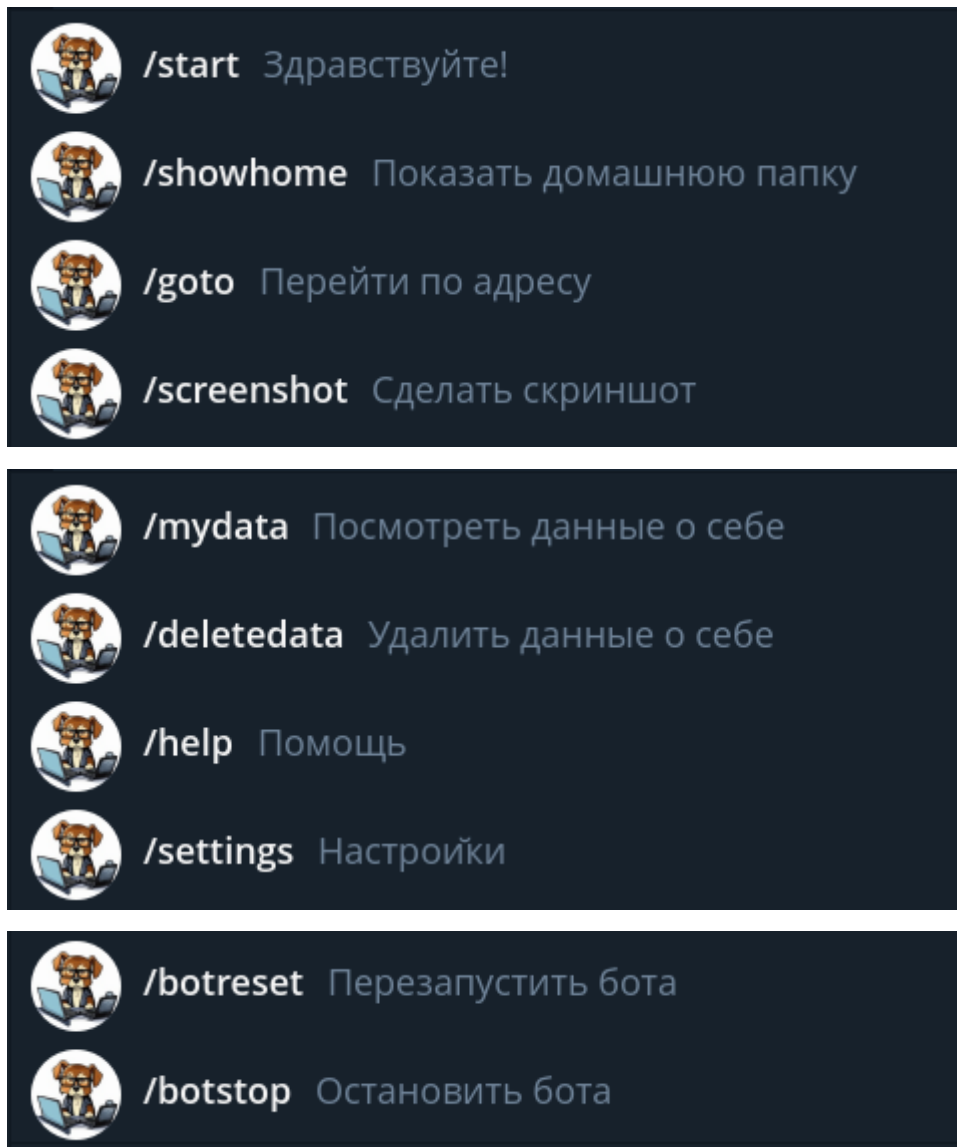
Telegram - это кроссплатформенный облачный мессенджер для мобильных устройств и компьютеров. Функциональность приложения может быть значительно расширена при помощи **ботов**. При помощи специального **API** сторонние разработчики могут создавать «ботов» - специальные аккаунты, управляемые программами. Типичные боты отвечают на специальные команды в персональных и групповых чатах, также они могут осуществлять поиск в интернете или выполнять иные задачи, применяются в развлекательных целях или в бизнесе. Скачать можно с официального сайта <https://telegram.org/>, описание API <https://core.telegram.org/api>.



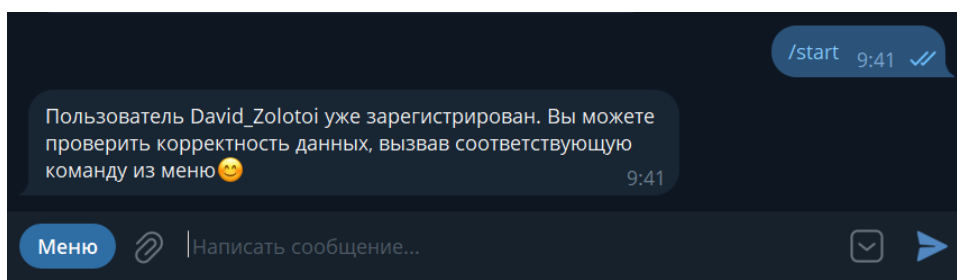
Глава 2. Задачи, описание и возможности чат-бота

Чат-бот обладает различным функционалом и элементами интерфейса, поддерживаемых ботами в Telegram. Имеет основное **меню** с командами, поддерживает работу с **Reply**-клавиатурами и с **Inline**-клавиатурами.

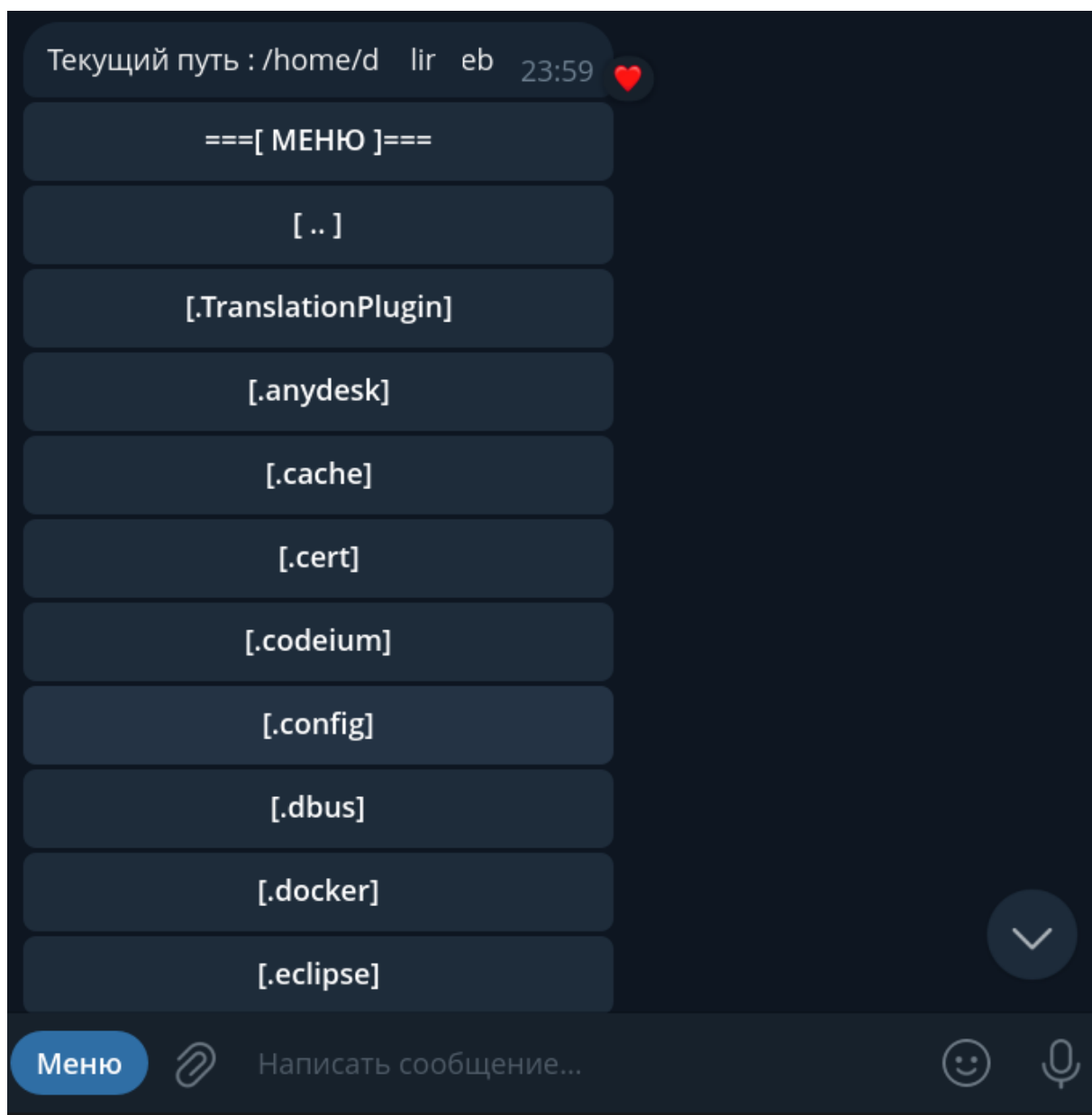
В основном меню находятся команды:



/start регистрирует и приветствует пользователя.



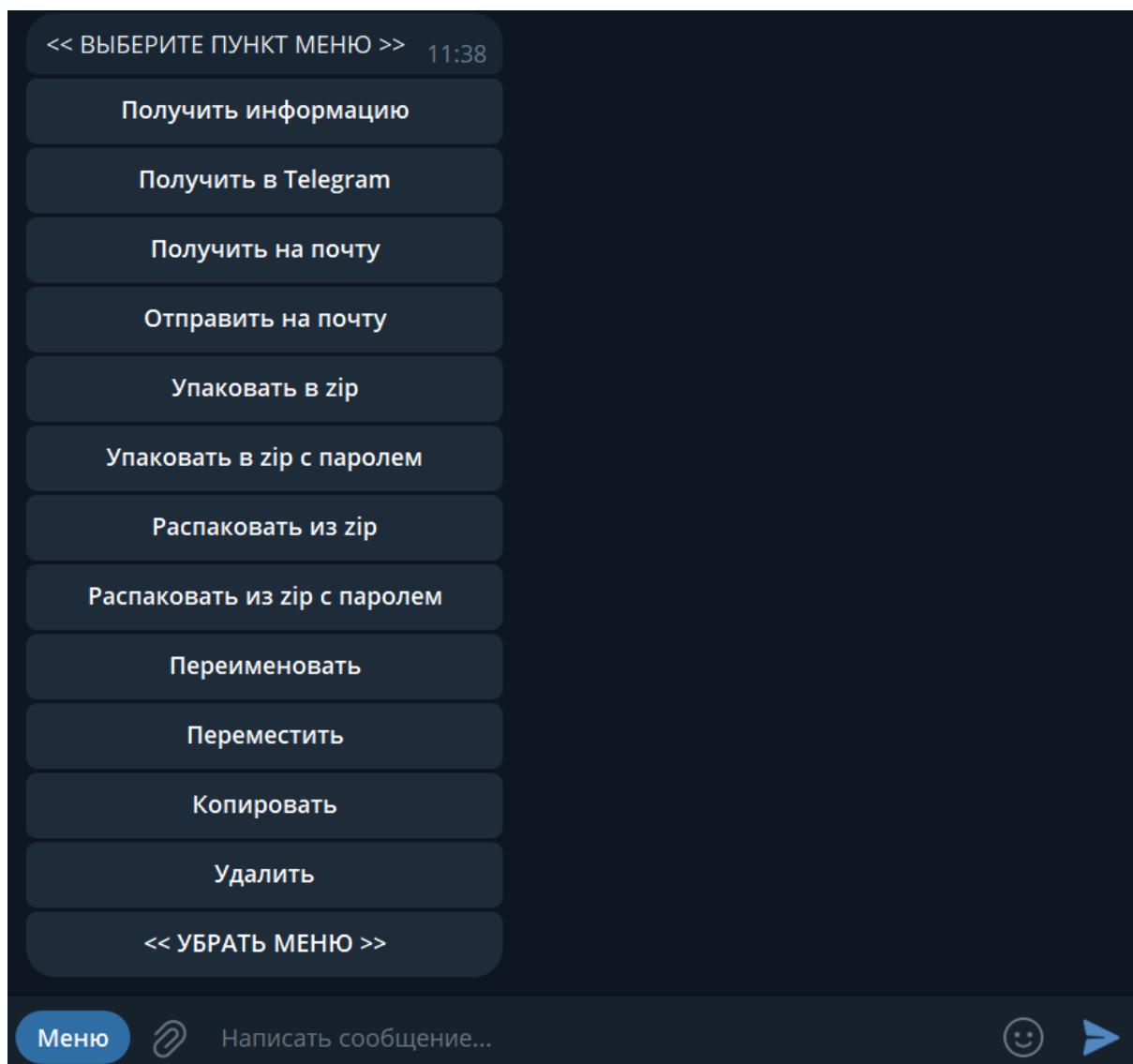
`/showhome` показывает в виде **Inline**-клавиатуры домашний каталог, являющийся таковым в используемой операционной системе. Используя Inline-кнопки, можно перемещаться по файловой системе и производить с её элементами различные действия. Посмотреть набор действий для выбора можно нажав на Inline-кнопку «`===[МЕНЮ]===`»:



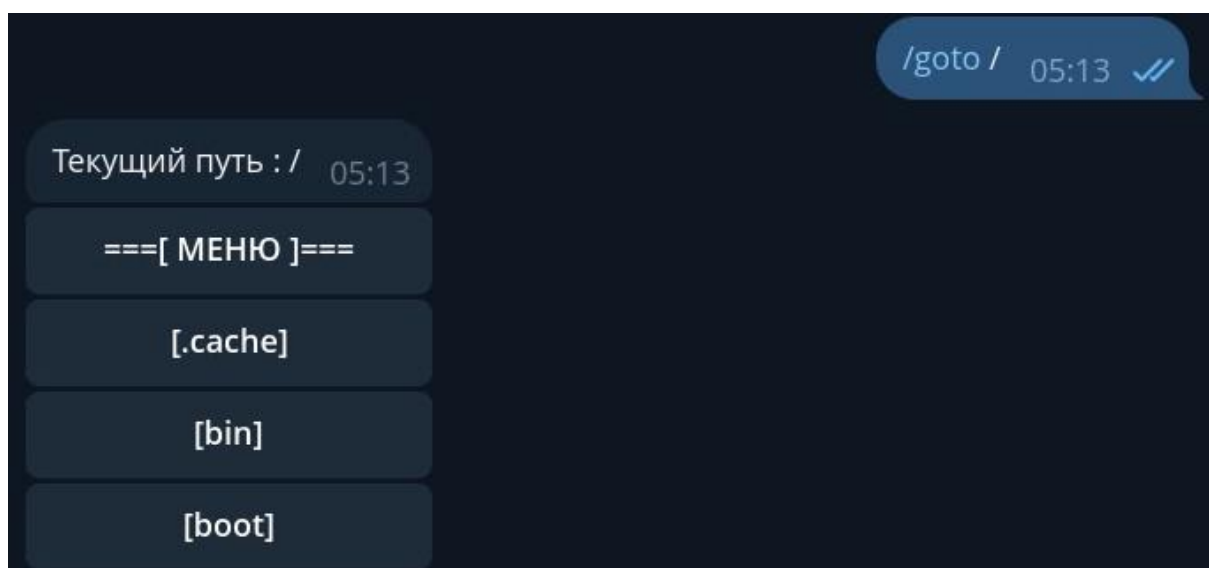
При нажатии на кнопку с названием папки, откроется содержимое папки.

При нажатии на кнопку с названием файла, откроется меню файла.

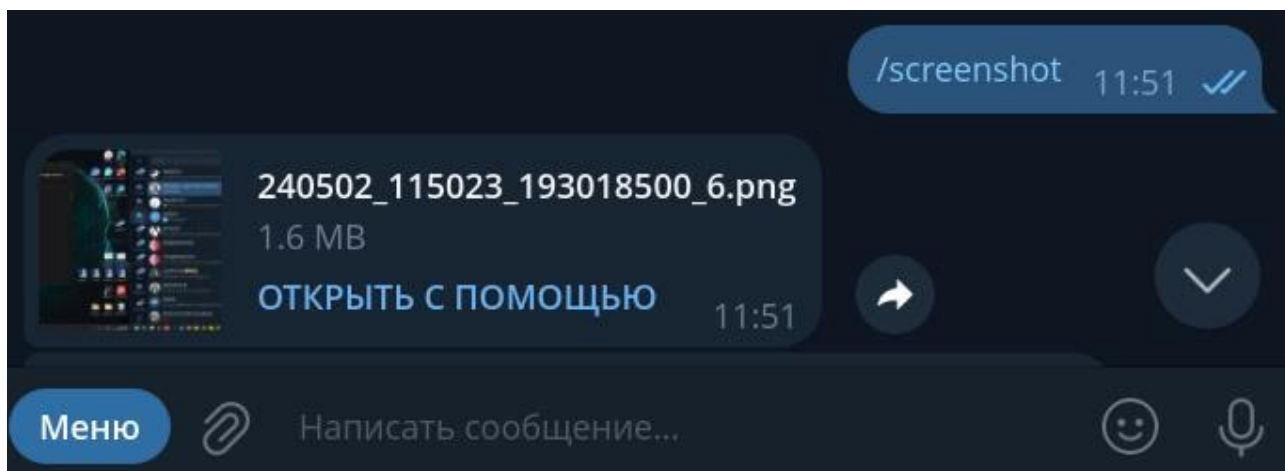
После нажатия на «**МЕНЮ**» появится набор возможных действий.



/goto переходит по конкретно указанному адресу, например в корень «/».

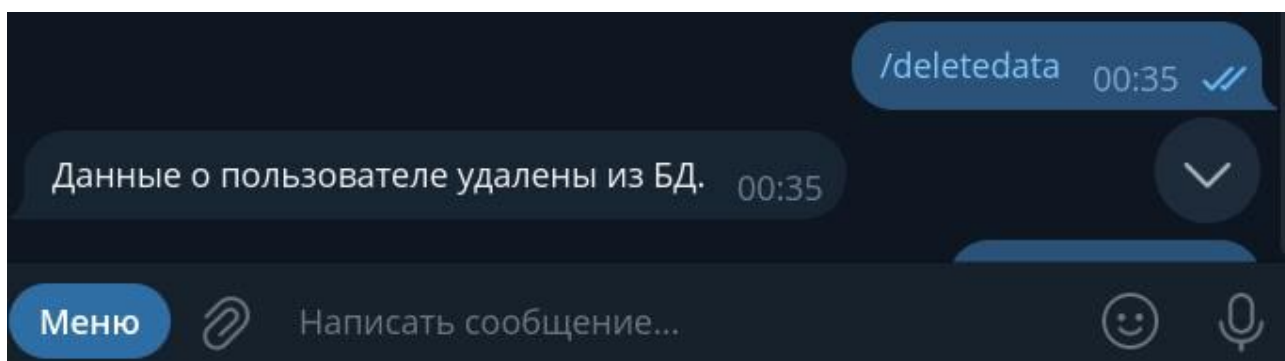


/screenshot делает скриншот на удаленном ПК, на нём же и сохраняет его, а потом прислает его в чат-бот.



/mydata показывает данные о пользователе, которые хранятся (например, настройки)

/deletedata удаляет пользователя из базы данных.



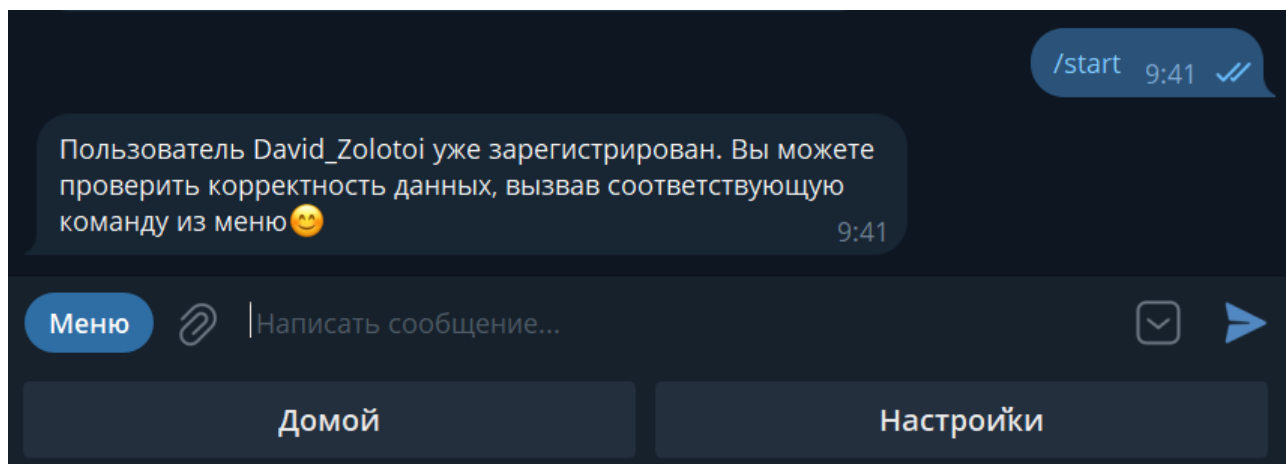
/help вызывает справку.

/settings показывает настройки, а также информацию о том, как их изменить.

/botreset перезапускает бота в отдельном процессе. После применения этой команды, окно с терминалом можно закрыть, программа будет продолжать работать.

/botstop останавливает бота на сервере. Последующий запуск уже невозможно будет сделать удаленно средствами Dogobot.

Под полем ввода можно вызвать **Reply**-клавиатуру, содержащую две кнопки: «Домой» и «Настройки».



Кнопка «Домой» выполняет те же функции, что и команда `/showhome` из меню.

Кнопка «Настройки» выполняет те же функции, что и команда `/settings` из меню.

Также есть команды с принимаемыми параметрами:

`/cp {путь}` скопирует текущий адрес (файл или папку) по указанному пути,

`/mv {путь}` переместит текущий адрес (файл или папку) по указанному пути,

`/rn {new name}` переименует текущий адрес (файл или папку) на новое имя,

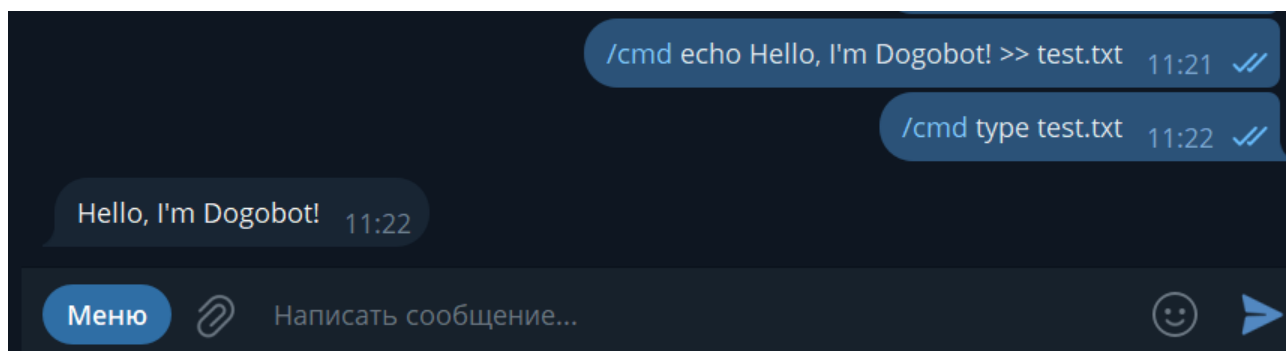
Команды `/setpass {новый пароль}`, `/setpmail {новый адрес личной почты}` и `/setomail {новый адрес другой почты}` устанавливают новые настройки пользователя: пароль упаковки/распаковки, адрес личной почты и другой почты.

`/cmd {CLI command}` запускает процесс, для выполнения команды в терминале.

*фигурные скобки в команде не указывать.

Применение команды `/cmd {CLI command}` может иметь серьезные последствия и рассчитано в первую очередь для опытных пользователей и администраторов.

Пример использования команды /cmd

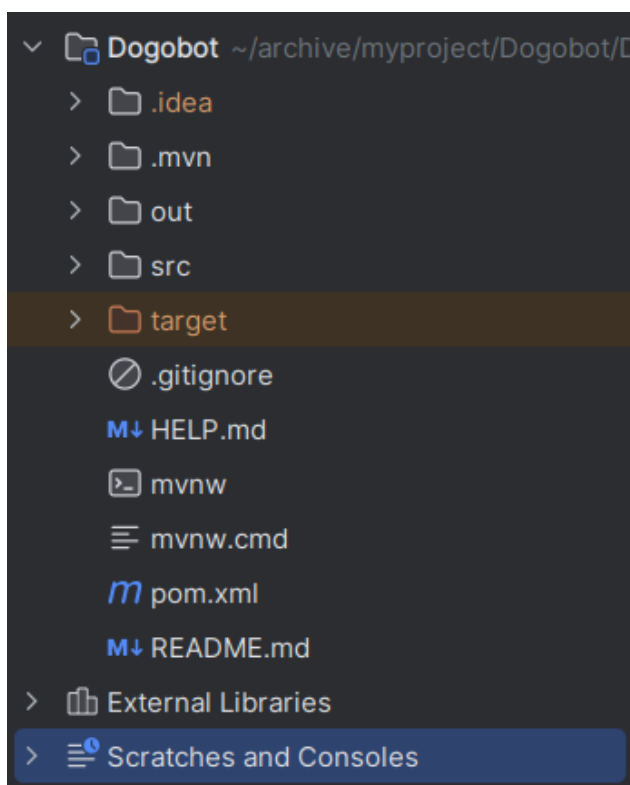


Иногда могут наблюдаться известные проблемы с кодировкой командной строки.

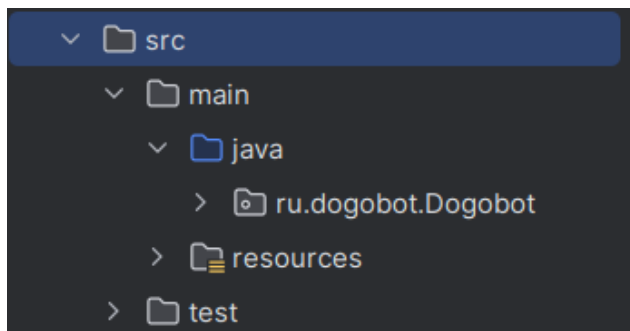
Глава 3. Структура программы

3.1 Общий состав проекта

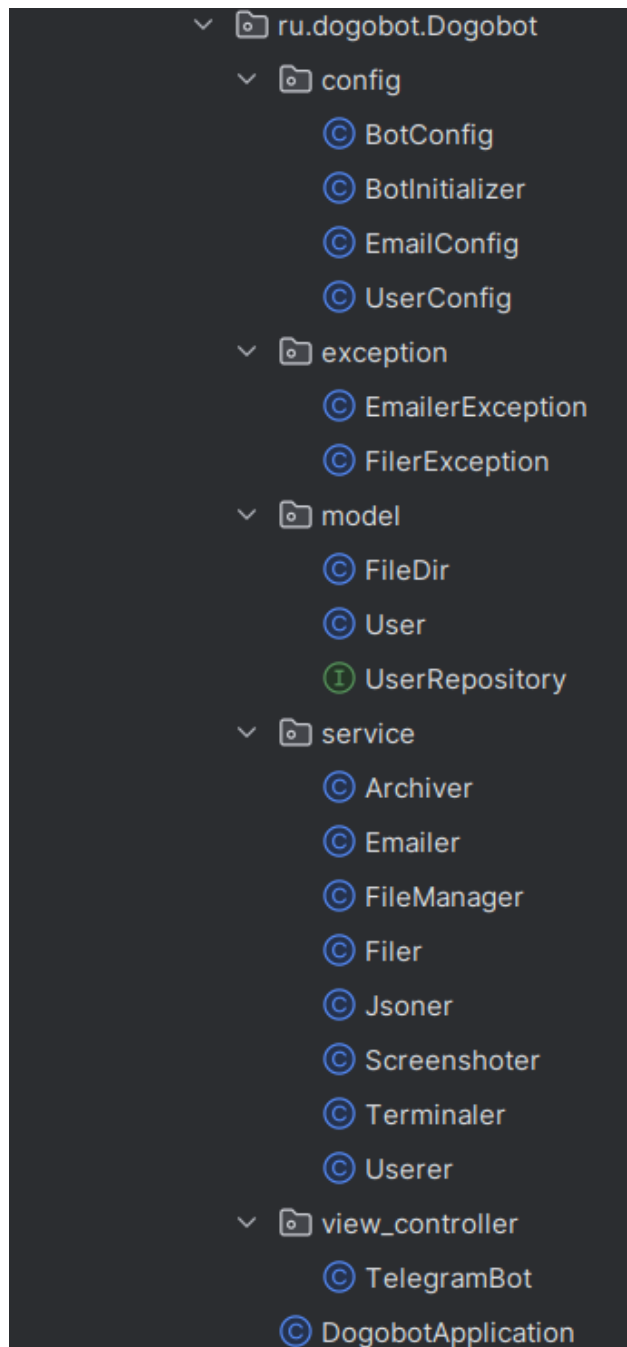
Проект был собран в Spring Initializr, с использованием Maven, с добавлением изменений, придерживающихся концепции MVC, но с некоторыми особенностями. Файл `pom.xml` и зависимости, указанные в нём, подробно описаны в следующих параграфах. В этом параграфе рассмотрим файловую структуру проекта, пакеты и основные классы, входящие в состав приложения. Корень проекта – обычный состав приложения, собранного с использованием Spring Initializr:



Папка `src` также особо ничем не выделяется от других проектов - только оригинальным названием проекта:



Содержимое папки resources описано в следующих параграфах, в этом параграфе рассмотрим пакеты и классы, содержащиеся в **ru.dogobot.Dogobot**:



ru.dogobot.Dogobot состоит из точки входа в классе DogobotApplication и пяти пакетов: model, view_controller, а также service, exception и config.

В пакете **view_controller** есть всего один класс – **TelegramBot**, который является как классом представления, так и контроллером. Класс TelegramBot наследуется от класса TelegramLongPollingBot из пакета org.telegram.telegrambots.bots относящемуся к Telegram Bot API и содержит

методы для взаимодействия с пользователем: как для получения информации от него, так и для отправки ему. В связи с чем, а также с учетом того, что проект маленький, было принято решение не разделять класс TelegramBot на два под-класса для View и для Controller. Но не исключено, что в будущем, с усложнением проекта это разделение будет реализовано.

Пакет **service** содержит в себе основную логику обработки информации. Главным классом в этом пакете является файловый менеджер **FileManager** – именно он первым получает запросы от TelegramBot. Остальные классы – это «инструменты» файлового менеджера, которые он использует для обработки информации. Работа файлового менеджера и такие функции, как навигация по файловой системе, получение информации о ф.п., переименование, перемещение, копирование, удаление реализуются в классах **FileManager** и **Filer**. Функции упаковки и распаковки ф.п., как с паролем, так и без него делегируются классу **Archiver**; работа со скриншотами – классу **Screenshoter**; работа с электронной почтой – классу **Emailer**; работу с пользовательскими данными (настройки и т.п.) – классу **Userer**; работу в терминале – классу **Terminaler**.

Для исключительных случаев работы сервисных классов создан пакет **exception** с «кастомными» исключениями **EmailerException** и **FilerException** наследующиеся от **Exception**.

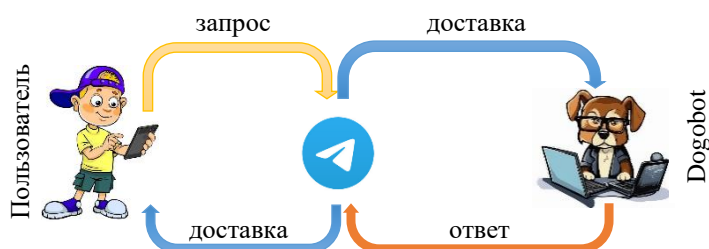
В качестве классов для пакета **model** выбраны класс **FileDir**, описывающий сущность элемента файловой системы (файл или папку) и класс **User** с данными о пользователе, синхронизирующийся с базой данных и т.п.

Пакет **config** содержит классы для настройки бота, почты и других личных настроек пользователя.

Каждый из классов подробно рассмотрен в следующих параграфах.

3.2 Алгоритм взаимодействия с пользователем в общих чертах

Простой сценарий эксплуатации заранее настроенного и запущенного чат-бота на своем устройстве (сервере):



1. У пользователя возникла потребность в функционале чат-бота, он открывает чат-бот в Telegram (неважно на каком устройстве) и инициирует запуск одной из функций для удовлетворения своей потребности;
2. Срабатывает Telegram API и передаёт соответствующую информацию о действиях пользователя запущенному на сервере приложению Dogobot;
3. Первым на сервере активируется экземпляр класса TelegramBot. После некоторой минимальной обработки информации, она передаётся одному из методов сервисного класса FileManager.
4. FileManager либо сам обрабатывает запрос (применяет необходимый метод для удовлетворения потребности пользователя), либо делегирует обработку одному из сервисных классов. Ответ после отработки метода, возвращается обратно в TelegramBot.
5. TelegramBot через Telegram API отправляет пользователю ответ.

3.3 Файл pom.xml и необходимые зависимости

Весь код файла pom.xml представлен в **приложении 20**.

В файле **pom.xml** содержится конфигурация проекта на языке Maven.

<modelVersion>4.0.0</modelVersion>: указывает версию формата файла pom.xml.

<parent>: определяет родительский проект, от которого наследуются некоторые настройки. В данном случае, родительский проект - это spring-boot-starter-parent с версией 3.2.4.

<groupId>ru.dogobot</groupId>: указывает идентификатор группы проекта. В данном случае, это ru.dogobot.

<artifactId>Dogobot</artifactId>: указывает идентификатор проекта. В данном случае, это Dogobot.

<version>0.0.1-SNAPSHOT</version>: указывает версию проекта. В данном случае, это 0.0.1-SNAPSHOT.

<name>Dogobot</name>: указывает название проекта. В данном случае, это Dogobot.

<description>Virtual friend assistant Dogobot for command execution </description>: Описывает проект.

<properties>: определяет свойства проекта, которые могут быть использованы в других частях pom.xml. В данном случае, указана версия Java (java.version) и версия библиотеки Telegram (telegram.version).

<dependencies>: содержит список зависимостей проекта. Каждая зависимость определена с помощью тега **<dependency>**. В данном случае, указаны зависимости для Spring Boot, Lombok, Telegram Bot API, Emoji Java, PostgreSQL, Spring Data JPA, Apache Commons Email, Zip4j, и Spring Boot Test.

Spring Boot - это фреймворк для разработки веб-приложений на языке Java. Он предоставляет удобные инструменты и конвенции для быстрой разработки приложений. В данном проекте используется spring-boot-starter, который включает в себя все необходимые зависимости для создания веб-приложения.

Lombok - это библиотека аннотаций для Java, которая упрощает разработку кода. Она позволяет автоматически генерировать код, например, для создания геттеров и сеттеров. В данном проекте используется lombok для упрощения работы с аннотациями и генерации кода.

Telegram Bot API - это библиотека для разработки Telegram-ботов. Telegram-боты - это программы, которые могут выполнять различные задачи в Telegram, такие как отправка сообщений, обработка команд и управление группами. В данном проекте используется telegrambots для создания Telegram-бота и работы с Telegram API.

Emoji Java - это библиотека для работы с эмодзи в Java. Она предоставляет удобные методы для работы с эмодзи, например, для преобразования текста с эмодзи в изображения или для поиска эмодзи по их названию. В данном проекте используется emoji-java для работы с эмодзи в приложении.

PostgreSQL - это система управления базами данных с открытым исходным кодом. В данном проекте используется spring-boot-starter-data-jpa, который включает в себя все необходимые зависимости для работы с PostgreSQL через **JPA (Java Persistence API)**. JPA - это стандартная спецификация для доступа к данным в Java.

Apache Commons Email - это библиотека для отправки электронных писем в Java. В данном проекте используется commons-email для отправки электронных писем.

Zip4j - это библиотека для работы с архивами ZIP в Java. В данном проекте используется zip4j для работы с ZIP-архивами.

Spring Boot Test - это библиотека для тестирования веб-приложений на Spring Boot. Она предоставляет удобные инструменты для тестирования контроллеров, сервисов и других компонентов приложения. В данном проекте используется spring-boot-starter-test для тестирования приложения.

3.4 Содержимое папки resources

Папка resources обычно служит для хранения ресурсов, таких как файлы конфигурации, статические файлы, сообщения об ошибках и другие ресурсы, которые могут потребоваться приложению. В данном проекте в папке resources хранятся:

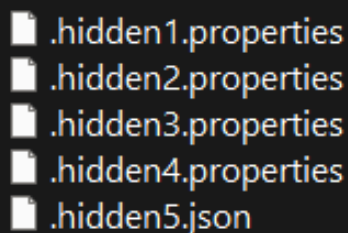
- файл конфигурации **application.properties**;
- файл конфигурации логов **logback.xml**;
- эмблема приложения – картинка **Dogobot.jpg**.

3.4.1 Файл конфигурации application.properties.

Содержимое:

```
spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden1.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden1.properties
```

В этом файле есть всего один параметр – это ссылка на другой файл конфигурации. Сделано это для того, чтобы вывести все файлы конфигурации и конфиденциальные данные из зоны действия git и корня приложения в специально заготовленную папку. Вторая закомментированная строка такая же, как и первая (созданная для работы в Linux), но для использования на другом ПК с установленной OS Windows. Файлы конфигурации разделены на 5 файлов по категориям и принадлежностям*: общие, Telegram, PostgreSQL, Email, личные данные (динамические, изменяемые удаленно). Помещены они в отдельную папку с названием «.hidden»:



```
.hidden1.properties
.hidden2.properties
.hidden3.properties
.hidden4.properties
.hidden5.json
```

Содержимое всех файлов конфигураций представлено в **приложении 22**.

*Примечание. Совсем необязательно так разделять, можно все параметры собрать в одном месте или разделить по-другому.

Файл **.hidden1.properties**, на который ссылается **application.properties** – это некоторые общие настройки без особой категории:

```
spring.application.name=[наименование приложения - Dogobot]
spring.main.headless=false

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden2.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden2.properties
```

Настройка **spring.main.headless=false** указывает на то, что при использовании приложения будут задействованы графические возможности – это необходимо, чтоб сервис Screenshoter мог исправно работать.

Файл **.hidden2.properties**, на который ссылается **.hidden1.properties** – это настройки, связанные с Telegram API:

```
telegrambot.name=botdogobot
telegrambot.token=[токен, полученный для бота]
telegrambot.authorId=[id автора программы]
telegrambot.ownerId=[id владельца программы]

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden3.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden3.properties
```

Информация о том, как получить эти данные описана в следующей главе.

Файл **.hidden3.properties**, на который ссылается **.hidden2.properties** – это настройки для работы с базой данных:

```
spring.datasource.url=[jdbc:postgresql-путь к базе данных]
spring.datasource.username=[username для входа в базу данных]
spring.datasource.password=[пароль для входа в базу данных]
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden4.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden4.properties
```

Эти настройки предоставляет администратор базы данных. Развернуть собственную базу данных можно:

- воспользовавшись услугами хостинга,
- воспользовавшись возможностями docker,
- локально, с помощью программ pgAdmin, dbeaver и настройкой доступа к ней извне.

Файл **.hidden4.properties**, на который ссылается **.hidden3.properties** – это на настройки для работы с электронной почтой:

```
email.smtp.host=[сервер исходящей почты]
email.smtp.port=[порт исходящей почты]
email.imap.host=[сервер входящей почты]
email.imap.port=[порт входящей почты]
email.from=[почтовый адрес с которого будут отправляться письма]
email.password=[пароль от почтового адреса]
```

Их можно получить у провайдера электронной почты.

Файл **.hidden5.json** – это файл, который никак не связан с файлом **application.properties** – это отдельный JSON-файл, содержащий некоторые настройки пользователя. На момент написания проекта – это: пароль для упаковки/распаковки файлов и папок; почта для получения файлов; другая почта, на которую пользователь может отправить файлы.

Для разворачивания приложения необязательно использовать именно такое разбиение на категории и файлы. Подробнее этот вопрос в следующей главе.

```
{
  "PACK_PASSWORD_KEY": "пароль для упаковки и распаковки файлов и папок",
  "PERSONAL_EMAIL_KEY": "адрес персональной почты",
  "OTHER_EMAIL_KEY": "дополнительный адрес другой почты"
}
```

Эти данные являются динамическими. Они также хранятся и в базе данных. Они задаются при первоначальной настройке и используются при запуске бота на сервере, но со временем пользователь может захотеть их поменять удаленно, и он сможет это сделать – данные обновятся, как в базе данных, так и в JSON -файле.

3.4.2 Файл конфигурации логов logback.xml

Весь код файла logback.xml представлен в **приложении 21**.

Файл **logback.xml** - это конфигурационный файл для библиотеки Logback, которая используется для логирования в Java-приложениях. В данном случае, файл logback.xml настроен для записи логов в файл dogo.log в папке «**../logs**», т.е. он также, как и файлы конфигураций вынесен из зоны действия git и корня приложения.

В конфигурации logback.xml определены следующие элементы:

<property name="HOME_LOG" value="..\logs\dogo.log"/>: Эта строка определяет свойство HOME_LOG, которое указывает на путь к файлу лога. Здесь указан относительный путь ../logs\dogo.log, что означает, что файл лога будет находиться в папке logs на один уровень выше текущего каталога.

<appender name="FILE-ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">: Эта строка определяет аппендер FILE-ROLLING, который используется для записи логов в файл dogo.log. Аппендер RollingFileAppender поддерживает ротацию файлов на основе размера и времени.

<file>\${HOME_LOG}</file>: Эта строка указывает путь к файлу лога, который определен в свойстве HOME_LOG.

3.4.3 Эмблема приложения.

Эмблема была сгенерирована в нейросетях. Символизирует преданную и умную собачку по кличке «**Dogobot**», умеющую и охотно выполняющую команды. Под лапами – два ПК, которые она символично связывает, сидит и ждёт команду.



Глава 4 Настройка, сборка и запуск проекта

Для работы приложения необходимо нужно настроить приложение, собрать его и запустить.

4.1 Файл – `application.properties`

Перед сборкой **необходимо и достаточно** настроить только один файл – `application.properties`.

На усмотрение администратора/пользователя его можно настроить по-разному. Самый простой способ – собрать абсолютно все параметры в одном файле `application.properties`. Другой способ – разбить параметры по категориям (файлам) так, как это сделано на момент написания работы и описано в пункте описания папки ресурсов **3.4.1***.

Главное – оставить неизменными наименования параметров и поменять только значения.

*Примечание. Ссылки в файле `application.properties` на импорт настроек из другого файла могут быть только абсолютными – этим объясняется потребность в сборке проекта после настройки. В будущем планируется создать отдельный микросервис (вероятно, с GUI интерфейсом, например с применением Swing), с помощью которого можно настраивать основное приложение.

4.1.1 Все параметры, которые необходимо настроить

- **spring.application.name**=[наименование приложения - Dogobot]
- **spring.main.headless**=false
- **telegrambot.name**=botdogobot
- **telegrambot.token**=[токен, полученный для бота]
- **telegrambot.authorId**=[id автора программы]
- **telegrambot.ownerId**=[id владельца программы]
- **spring.datasource.url**=[jdbc:postgresql-путь к базе данных]
- **spring.datasource.username**=[username для входа в базу данных]
- **spring.datasource.password**=[пароль для входа в базу данных]
- **spring.datasource.driver-class-name**=org.postgresql.Driver
- **spring.jpa.hibernate.ddl-auto**=update
- **spring.jpa.show-sql**=true
- **email.smtp.host**=[сервер исходящей почты]
- **email.smtp.port**=[порт исходящей почты]
- **email.imap.host**=[сервер входящей почты]
- **email.imap.port**=[порт входящей почты]
- **email.from**=[почтовый адрес с которого будут отправляться письма]
- **email.password**=[пароль от почтового адреса]

Параметр **spring.main.headless=false** указывает на то, что при использовании приложения будут задействованы графические возможности — это необходимо, чтоб сервис Screenshoter мог исправно работать.

4.1.2 Параметры telegrambot.

На момент написания проекта, приложение работает только с оригинальным токеном, а значит каждому пользователю необходимо будет получить свой токен для работы приложения. Токен и наименование бота необходимо получить по инструкции с официального сайта <https://core.telegram.org/api>. Вкратце, есть бот <https://t.me/BotFather>, имеющий возможность выделять для нового бота токен. Следуя его инструкциям, можно зарегистрировать нового бота (с новым наименованием) и управлять некоторыми его составными частями, например описанием, аватаром и т.п. В будущем планируется свести работу бота для всех пользователей к одному токену.

`telegrambot.authorId` и `telegrambot.ownerId` – это Id пользователей, которые должны будут получать сообщения от бота. Обычные сообщения во время использования чат-бота будут отправляться на `ownerId`, административные сообщения (касающиеся работы приложения и технических сбоев) будут присылаться администратору приложения. Это может быть один и тот пользователь.

4.1.3 Параметры `spring.datasource` и `spring.jpa`

Это параметры, необходимые для корректной работы с базой данных и её управлением из Spring-приложения.

Эти настройки предоставляет администратор базы данных. Развернуть собственную базы данных можно:

- воспользовавшись услугами хостинга и их инструкциями,
- воспользовавшись возможностями `docker`,
- локально, с помощью программ `pgAdmin`, `dbeaver` и настройкой доступа к ней извне.

4.1.4 Параметры email

Это параметры, необходимые для корректной работы с электронной почтой. Их необходимо получить у провайдера электронной почты.

Например, в случае почтой mail.ru на домене @bk.ru корректно работают следующие настройки:

- email.smtp.host=smtp.mail.ru
- email.smtp.port=465
- email.imap.host=imap.mail.ru
- email.imap.port=993

Современные технологии не позволяют использовать в приложениях основной пароль. Для использования почты в приложениях используются специально сгенерированные в веб-интерфейсе провайдера пароли электронной почты.

4.2 Файл JSON с пользовательскими настройками

Эти данные являются динамическими. Они также хранятся и в базе данных. Они задаются при первоначальной настройке и используются при запуске бота на сервере, но со временем пользователь может захотеть их поменять удаленно, и он сможет это сделать через бот – данные обновятся, как в базе данных, так и в JSON -файле.

На момент написания проекта – это: пароль для упаковки/распаковки файлов и папок; почта для получения файлов; другая почта, на которую пользователь может отправить файлы.

Путь к JSON-файлу настроек записан в классе **UserConfig** в константе **FILE_PATH**.

```
{  
    "PACK_PASSWORD_KEY": "пароль для упаковки и распаковки файлов  
и папок",  
    "PERSONAL_EMAIL_KEY": "адрес персональной почты",  
    "OTHER_EMAIL_KEY": "дополнительный адрес другой почты"  
}
```

4.3 Пути к логам и скриншотам

В файле `logback.xml`, находящемся внутри папки ресурсов можно настроить параметры логирования, в том числе и путь к папке, в которой будут создаваться, архивироваться и храниться логи.

Путь указывается в теге (может быть, как абсолютным, так и относительным) `<property name="HOME_LOG" value="..\logs\dogo.log"/>`

В данном случае логи будут храниться в папке `logs`, находящейся рядом с папкой, содержащей папку исполняемого файла приложения.

Подробнее о других тегах настройки логирования описано в пункте **3.4.2**.

Если есть необходимость настроить путь к папке для сохранения скриншотов, то это возможно сделать в файле класса **Screenshoter** в строке

```
String screenshotDirPath = "../screenshots";
```

В данном случае скриншоты будут храниться в папке `screenshots`, находящейся рядом с папкой, содержащей папку исполняемого файла приложения.

Путь может быть, как абсолютным, так и относительным.

4.4 Сборка проекта и его запуск.

4.4.1 Сборка проекта

Осуществляется с помощью **Apache Maven** - это инструмент для управления проектами и сборки программного обеспечения на основе концепции управления зависимостями. Он позволяет автоматизировать процесс сборки проекта, управлять зависимостями, выполнять тестирование и многое другое. Скачать можно с оф. сайта - <https://maven.apache.org/>.

После установки нужно в терминале открыть корень папки проекта и в нём запустить команду «**mvn clean && mvn compile && mvn package**»:

- **mvn clean** очищает временные файлы и папки, созданные во время сборки проекта. Она удаляет целевую папку, в которой хранятся скомпилированные классы и другие временные файлы.
- **mvn compile** компилирует исходный код проекта. Она компилирует Java-файлы в скомпилированные классы, которые могут быть запущены на виртуальной машине Java (JVM).
- **mvn package** создает упаковку проекта. Она выполняет все предыдущие шаги (clean, compile) и добавляет к ним другие шаги, такие как сборка ресурсов, создание JAR-файла или WAR-файла, и добавление всех необходимых зависимостей в упаковку. Получается готовая для развертывания упаковка проекта.

В результате последовательного выполнения этих команд, создаётся готовая для развертывания упаковка проекта, в том числе в корне проекта в папке target появляется/обновляется файл jar.

Например, **Dogobot-0.0.1-SNAPSHOT.jar**.

4.4.2 Запуск проекта

Так как **Dogobot-0.0.1-SNAPSHOT.jar** – это java-приложение, то для его запуска необходима установленная Java, а точнее:

- **Java Development Kit (сокр. JDK)**, который можно скачать с официального сайта <https://www.oracle.com/java/technologies/downloads/>;

или

- **OpenJDK** (проект по созданию полностью совместимого Java Development Kit, состоящего исключительно из свободного и открытого исходного кода), который можно скачать с официального сайта <https://jdk.java.net/>.

Как видно из файла `pom.xml` для данного проекта необходима Java 17-й версии.

Если пути к логам и скриншотам настроены так, как предлагалось выше, то перед запуском проекта рекомендуется:

1. Выбрать место на диске и создать в нём папку, например с названием «**Dogobot**»;
2. В папке «**Dogobot**», созданной пунктом выше, создать еще одну папку, например «**jar**»
3. В папку «**jar**», созданной пунктом выше, поместить собранный проект **Dogobot-0.0.1-SNAPSHOT.jar**

Теперь само приложение находится в папке по адресу “.../Dogobot/jar/”, логи будут записаны в папку по адресу “.../Dogobot/logs/”, а скриншоты в папку по адресу “.../Dogobot/screenshots/”.

Для запуска приложения необходимо:

1. Открыть в терминале папку с файлом `jar`, в случае, описанном выше – это путь “.../Dogobot/jar/”;
2. Запустить команду: **java -jar Dogobot-0.0.1-SNAPSHOT.jar**.

При желании из приложения можно сделать сервис, работающий в фоне («демон»). Вот примерная настройка для таких целей:

```
[Unit]
Description=Virtual friend assistant Dogobot for command execution
After=syslog.target network.target

[Service]
SuccessExitStatus=143

User=root
Group=root

Type=simple

ExecStart=java -jar /opt/bots/tg/Dogobot.jar

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

# работа с сервисами
systemctl status [serviceName]
systemctl start [serviceName]

# запуск файла исполняемого файла jar:
java -jar [путь]
```


Глава 5 Подробнее о классах

5.1 Класс TelegramBot

5.1.1 Общая информация

Полный код класса находится в **приложении 19**.

Класс **TelegramBot** наследуется от **TelegramLongPollingBot**, содержит фильтр пользователей, от которых могут поступать запросы, логику делегирования запросов, прилетающих от пользователя, логику формирования меню, Reply- и Inline-клавиатур, отправки сообщений, их изменения и удаления.

Аннотация **@Service** над классом используется для обозначения компонента, который предоставляет бизнес-логику или сервис.

Код класса разделен на части «**region**»:

- «**КОНСТАНТЫ** и другие исходные данные» содержит поле **botConfig** для настройки бота; набор именованных констант **BotMenuEnum** для основного меню бота и лист команд на основе него **botMenu**; набор именованных констант **ReplyKeyboardEnum** для наименований кнопок Reply-клавиатуры под полем ввода в чат-боте и лист наименований **replyKeyboardNames** на его основе; набор именованных констант **OtherCommandEnum** для команд со слешем, но не являющихся пунктами меню; **Id** последнего сообщения **fileDirMessageId**.
- «**AUTOWIRED and CONSTRUCTORS**» содержит внедрение зависимостей: **FileManager** через аннотацию **@Autowired**, а конфигурация бота - через конструктор. В конструкторе также происходит построение основного меню.
- «**OVERRIDE TelegramBots + фильтр на владельца**» – содержит методы от суперкласса, которые должны быть переопределены (**getBotUsername()**, **getBotToken()**, **onUpdateReceived(Update update)**), а также метод **ownerFilter(Update update)** для фильтрации и игнорирования всех пользователей, которые не имеют отношения к работе приложения. Основное распределение реального запроса для его дальнейшей обработки происходит в методе **onUpdateReceived(Update update)** – о нём более подробно расписано ниже.

- «Обработчики команд со слешем» и
- «Обработчики конкретных команд со слешем» распознают в запросе команду со слешем методом `boolean isCommand(..)` и передают её в обработку определённому методу из файлового менеджера.
- «Обработчики Callback» содержит минимальную логику навигации по файловой системе, распознаёт Callback-команду методом `boolean isCallbackData(..)` и обрабатывает её.
- «ОТПРАВКА И ИЗМЕНЕНИЕ СООБЩЕНИЙ» содержит различные вариации методов для отправки сообщений: с Inline- или Reply-клавиатурами и без них, отправки файла, изменения или удаления сообщения.
- «КЛАВИАТУРЫ» содержит методы создания и возврата Inline- или Reply-клавиатур на основе исходных данных, например таблиц (лист листов) имен.

5.1.2 «Траектория движения запроса»

Любой запрос, посланный в Telegram боту попадает в метод `onUpdateReceived(Update)`, вот его код:

```
@Override
public void onUpdateReceived(Update update) {
    if (!ownerFilter(update)) return;
    if (update.hasMessage() && update.getMessage().hasText()) {
        //Если прилетел текст
        handlerText(update);
    } else if (update.hasCallbackQuery()) {
        //Если прилетел CallbackQuery
        handlerCallbackQuery(update);
        //todo можно добавить сохранение присланных файлов, но
        //придется для каждого типа файла (музыка, видео и т.п.) делать от-
        //дельный метод
    } else {
        log.error("Не могу распознать отправленную информацию: " +
            update);
    }
}
```

В этом методе сначала происходит фильтрация, т.е. Id пользователя, который прислал запрос не подходит, то на этом всё заканчивается.

Затем проверяются данные объекта обновления `update` для определения типа данных в нём.

Если суть запроса заключается в простом **тексте**, то его обработка передаётся в метод **`handlerText(update)`**, если в запросе есть **CallBack**-команда, то его обработка передаётся в метод **`handlerCallbackQuery(update)`**, иначе записывается лог о том, что запрос никак не распознан. В планах по улучшению проекта есть обработка поступающих файлов.

В методе `handlerText(update)` полученный запрос сопоставляется с известными программе командами из именованных списков констант `BotMenuEnum` и `OtherCommandEnum`. Если есть совпадение, то запрос передаётся на обработку в соответствующий метод файлового менеджера.

В методе `handlerCallbackQuery(update)` полученный запрос проверяется является ли нажатая Inline-кнопка кнопкой для навигации по файловой системе.

Если нет, то запрос сопоставляется с известными программе Callback-командами из именованного списка констант `FileManager.FileDirMenu`. Если есть совпадение, то запрос передаётся на обработку в соответствующий метод файлового менеджера.

В планах по улучшению проекта есть разбиение файла на большее количество файлов.

5.2 Классы **FileManager**, **FileDir** и др.

Полный код класса **FileManager** находится в **приложении 13**, а класса **FileDir** - в **приложении 8**.


Экземпляр класса **FileManager** является тем, кто первым принимает запрос от TelegramBot для дальнейшей обработки. И **FileManager** и **TelegramBot** в импортах имеют класс **FileDir**, который представляет сущность элемента файловой системы (файла или папки).

Аннотация **@Service** над классом **FileManager** используется для обозначения компонента, который предоставляет бизнес-логику или сервис.

Содержит:

- Логика работы с экземпляром класса **FileDir** – создание и заполнение.
- Набор именованных констант **FileDirMenu** для меню возможных действий с файлом или папкой (Inline-кнопками); через аннотацию **@Autowired** добавлены все сервисные классы; словарь для актуального пути **currentPathDict** со входящими в его состав элементами файловой системы **FileDir**.

• «region РАБОТА С ПОЛЬЗОВАТЕЛЯМИ» содержит метод получения **Id** из объекта обновления **Long getChatIdFromUpdate(Update update)**, причем неважно какого рода запрос прилетел; методы работы с базой данных – поиска, регистрации, обновления личных настроек пользователя, удаления данных о пользователе. Обработка запросов из этих методов будет делегирована сервисному классу **Userer**, который в свою очередь работает с репозиторием **UserRepository**. **UserRepository**, является интерфейсом, расширяющим интерфейс **JpaRepository<User, Long>** - это применение **org.springframework.data.jpa** в работе с базой данных. Моделью для сопоставления с таблицей в базе данных является класс **User** из пакета **model**, содержащий данные о пользователе, в том числе и его настройки. Полный код класса **Userer** находится в **приложении 18**, а класса **User** – в **приложении 09**.



users_data_table	
123	chat_id
ABC	first_name
ABC	last_name
🕒	registered_at
ABC	user_name
ABC	other_email
ABC	pack_password
ABC	personal_email

- Метод для обработки запроса по созданию и получению скриншота **printScreen()**. Обработка запросов из этого метода будет делегирована сервисному классу **Scerenshoter**, который содержит в себе логику и методы для создания скриншотов. Полный код класса Scerenshoter находится в **приложении 16**.

- «region РАБОТА С ЭЛЕКТРОННОЙ ПОЧТОЙ» содержит методы обработки запроса для отправок писем по электронной почте. Обработка запросов из этого метода будет делегирована сервисному классу **Emailer**, который содержит логику и методы для отправки писем с вложением и без, а также простейшую валидацию адреса получателя с помощью регулярных выражений. Полный код класса Emailer находится в **приложении 12**.

- «region РАБОТА С АРХИВАМИ» содержит методы обработки запроса для упаковки/распаковки файлов и папок, как с паролем, так и без него. Обработка запросов из этого метода будет делегирована сервисному классу **Archiver**, который содержит логику и методы для упаковки/распаковки файлов и папок, как с паролем, так и без него. Полный код класса Archiver находится в **приложении 11**.

- «region РАБОТА С ФАЙЛАМИ» содержит методы обработки запросов переименования, перемещения, копирования, удаления. Обработка запросов из этого метода будет делегирована сервисному классу **Filer**, который содержит логику и методы для переименования, перемещения, копирования, удаления элементов файловой системы. Полный код класса Filer находится в **приложении 14**.

- Два метода для работы с терминалом: метод **terminalExecute(String script)** для выполнения любой команды в терминале и метод **String botReset()** для удаленной перезагрузки бота с использованием терминала. Обработка запросов из этого метода будет делегирована сервисному классу **Terminaler**, который содержит логику и методы для работы с терминалом. Полный код класса Terminaler находится в **приложении 17**.

Почти вся логика и методы класса FileManager заключены в блоки try/catch и отлавливают исключения до того, как вернут в TelegramBot обработанную

информацию (отчет в виде текста или файл), что приводит к минимуму вероятность падения приложения. В пакете exception созданы два типа исключений: **EmailerException** и **MailerException** для исключений во время работы с электронной почтой и работы с файловыми операциями соответственно. Полный код класса EmailerException находится в **приложении 06**, а класса MailerException - **приложении 07**.

На случай некорректной работы бота есть команда удаленной перезагрузки бота, которая создаёт новый процесс с запущенным ботом, а работу текущего экземпляра завершает.

5.3 Классы пакета config

Это конфигурационные классы и помечаются они аннотацией @Configuration.

Их четыре:

- **BotConfig** хранит настройки бота: токен, имя, Id автора и владельца.
- **BotInitializer** содержит один единственный метод, который при запуске приложения инициализирует бота.
- **EmailConfig** хранит настройки электронной почты: адреса серверов для входящей и исходящей почты, порты для них, логин и пароль.
- **UserConfig** хранит динамические данные — личные данные пользователя, которые он может менять удаленно: пароль упаковки/распаковки, персональный адрес и другой адрес электронной почты. Эти данные хранятся, как в базе данных, так и в JSON-файле настроек. Для управления JSON-файлом настроек создан сервисный класс **Jsoner**, который добавлен в UserConfig через аннотацию @Autowired.

Jsoner содержит методы прочтения и добавления в JSON-файл пар «ключ-значение».

Полный код конфигурационных классов пакета config находится в приложениях 02, 03, 04, 05, а код сервисного класса Jsoner — в приложении 15.

Заключение

Данный проект – разработка полезной программы для пользователей, которая удовлетворяет определенный набор потребностей взамен использования известных и мощных программ для удаленного доступа.

Проект можно развивать, увеличивать его функционал до невообразимых на сегодня результатов. Присутствует и кроссплатформенность – на момент написания проекта, работа проверена на OS Linux Debian и OS Windows (10 Pro и 11 Pro).

В проекте использованы популярные и современные технологии: IntelliJ IDEA, Java Development Kit, Spring Initializr (SpringBoot), Apache Maven, DBeaver (PostgreSQL), Git и GitHub, Telegram API.

В проекте используются: объектно-ориентированное программирование, навыки работы со Spring, сборкой проекта инструментом Maven с подходом MVC, системой контроля версий, использованием принципов SOLID и паттернов проектирования для обеспечения гибкости и расширяемости кода.

Список используемой литературы

- «Spring Boot в действии» Крейг Уолш, Джош Лонг.
- «Spring in Action» Крейг Уолш.
- «Maven: The Definitive Guide» Соня Хэyder, Тим О'Брайен.
- «Maven: The Complete Reference» Тим О'Брайен
- «Pro Git» Скотт Чакон, Бен Линус.
- «Git Pocket Guide» Ричард Сильверман
- «IntelliJ IDEA Essentials» Jarosław Krochmalski.
- «Java: Полное руководство» Герберт Шилдт.
- «Java: A Beginner's Guide» Герберт Шилдт.
- «PostgreSQL: Up and Running» Регина О'Рейли

Веб-источники.

- Java и JDK <https://www.oracle.com/cis/java> и <https://jdk.java.net>.
- IntelliJ IDEA <https://www.jetbrains.com/ru-ru/idea>.
- Telegram Bot API <https://telegram.org> и <https://core.telegram.org/api>.
- Spring <https://start.spring.io>.
- PostgreSQL <https://www.postgresql.org> и <https://dbeaver.io>.
- GitHub Guides <https://github.com> и <https://git-scm.com/book/ru/v2>.
- Apache Maven <https://maven.apache.org>.
- Энциклопедия <https://ru.wikipedia.org>

Приложения

Приложение 01. Содержимое класса DogobotApplication

```
package ru.dogobot.Dogobot;

import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@Slf4j
@SpringBootApplication
public class DogobotApplication {

    public static void main(String[] args) {
        log.info(
            System.lineSeparator() + System.lineSeparator()
            + "#####" +
System.lineSeparator()
            + "##### START #####" +
System.lineSeparator()
            + "#####"
            + System.lineSeparator()
        );
        SpringApplication.run(DogobotApplication.class, args);
    }
}
```

Приложение 02. Пакет config, класс BotConfig

```
package ru.dogobot.Dogobot.config;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Slf4j
@Configuration
@EnableScheduling
@Data
public class BotConfig {

    @Value("${telegrambot.token}")
    String token;

    @Value("${telegrambot.name}")
    String botName;

    @Value("${telegrambot.authorId}")
    Long authorId;

    @Value("${telegrambot.ownerId}")
    Long ownerId;

}
```

Приложение 03. Пакет config, класс BotInitializer

```
package ru.dogobot.Dogobot.config;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.event.ContextRefreshedEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.telegram.telegrambots.meta.TelegramBotsApi;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import org.telegram.telegrambots.updatesreceivers.DefaultBotSession;
import ru.dogobot.Dogobot.view_controller.TelegramBot;
@Slf4j
@Component
public class BotInitializer {
    @Autowired
    TelegramBot bot;
    /**
     * Инициализация бота при запуске
     * @throws TelegramApiException если возникли проблемы
     */
    @EventListener({ContextRefreshedEvent.class})
    public void init() throws TelegramApiException {
        TelegramBotsApi telegramBotsApi = new TelegramBotsApi(DefaultBotSession.class);
        try {
            telegramBotsApi.registerBot(bot);
            log.info("Бот успешно инициализирован");
        }
        catch (TelegramApiException e) {
            log.error("Проблемы при инициализации бота: " + e.getMessage());
        }
    }
}
```

Приложение 04. Пакет config, класс EmailConfig

```
package ru.dogobot.Dogobot.config;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;

@Slf4j
@Configuration
@Data
public class EmailConfig {

    @Value("${email.smtp.host}")
    private String smtpHost;

    @Value("${email.smtp.port}")
    private int smtpPort;

    @Value("${email.imap.host}")
    private String imapHost;

    @Value("${email.imap.port}")
    private int imapPort;

    @Value("${email.from}")
    private String emailFrom;

    @Value("${email.password}")
    private String password;

}
```

Приложение 05. Пакет config, класс UserConfig

```
package ru.dogobot.Dogobot.config;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import ru.dogobot.Dogobot.service.Jsoner;

import java.util.HashMap;
import java.util.Map;

@Slf4j
@Configuration
@Data
public class UserConfig {

    private Jsoner jsoner;
    final String FILE_PATH = "../.hidden/.hidden5.json";
    final int INDENT_FACTOR = 2;
    private Map<String, String> configs;

    //region Изменяемые параметры пользователя, которые хранятся не только в БД, но в JSON

    private final String PACK_PASSWORD_KEY = "PACK_PASSWORD_KEY";
    private String packPassword;

    private final String PERSONAL_EMAIL_KEY = "PERSONAL_EMAIL_KEY";
    private String personalEmail;

    private final String OTHER_EMAIL_KEY = "OTHER_EMAIL_KEY";
    private String otherEmail;
```

```

//endregion

@Autowired
public UserConfig(Jsoner jsoner) {
    this.jsoner = jsoner;
    this.configs = readConfigs(this.configs, PACK_PASSWORD_KEY, PERSONAL_EMAIL_KEY,
OTHER_EMAIL_KEY);

    if (!this.configs.containsKey(PACK_PASSWORD_KEY)) {
        log.warn("В словаре отсутствует пароль для упаковки/распаковки.");
    }
    this.packPassword = this.configs.get(PACK_PASSWORD_KEY);

    if (!this.configs.containsKey(PERSONAL_EMAIL_KEY)) {
        log.warn("В словаре отсутствует персональная почта.");
    }
    this.personalEmail = this.configs.get(PERSONAL_EMAIL_KEY);

    if (!this.configs.containsKey(OTHER_EMAIL_KEY)) {
        log.warn("В словаре отсутствует другая почта.");
    }
    this.otherEmail = this.configs.get(OTHER_EMAIL_KEY);
}

/**
 * Проверяет существование словаря настроек и ключей в нём.
 * Если словаря нет, то создаёт его.
 * Если ключей в словаре нет, то добавляет их, прочитав значения из файла.
 * @param configs словарь настроек
 * @param jsonKeys ключи, которые необходимо проверить или добавить
 * @return Проверенный и дополненный, в случае необходимости, словарь.
 */
public Map<String, String> readConfigs(Map<String, String> configs, String... jsonKeys) {

```



```

        if (configs == null) configs = new HashMap<>();
        for (var jsonKey:jsonKeys) {
            if (jsonKey == null) continue;
            if (!configs.containsKey(jsonKey)) {
                configs.put(jsonKey, jsoner.readJSONFile(FILE_PATH, jsonKey));
            }
        }
        return configs;
    }

    /**
     * Обновляет пару ключ-значение в словаре настроек и в JSON-файле
     * @param key ключ для обновления
     * @param value значение для обновления
     * @return результат изменения
     */
    public boolean updateMapAndJSON(String key, String value) {
        if (key == null) return false;
        boolean result = false;
        try {
            this.configs.put(key, value);
            result = this.jsoner.updateValueJSONFile(this.FILE_PATH, this.INDENT_FACTOR, key, value);
            log.info("Пара ключ-значение по ключу '%s' обновлена в словаре настроек и в JSON-файле.".formatted(key));
        } catch (Exception e) {
            log.warn("Не удалось обновить/добавить пару ключ-значение по ключу '%s' в словарь или в JSON-файл.%s%s"
                    .formatted(key, System.lineSeparator(), e.getMessage()));
        }
        return result;
    }

    /**
     * Обновляет пару ключ-значение для пароля упаковки/распаковки в конфигурации (поле, словарь

```

```

настроек и JSON-файле)
    * @param packPassword новое значение пароля
    * @return результат изменения
    */
    public boolean updateConfigPackPassword(String packPassword) {
        this.packPassword = packPassword;
        log.info("Поле packPassword экземпляра UserConfig обновлено.");
        boolean isUpdateMapAndJSON = this.updateMapAndJSON(PACK_PASSWORD_KEY, packPassword);
        if (!isUpdateMapAndJSON){
            log.warn("Не удалось обновить пароль пользователя в конфигурациях: словаре и/или JSON-
файле.");
        }
        return isUpdateMapAndJSON;
    }

    /**
     * Обновляет пару ключ-значение для персональной почты в конфигурации (поле, словарь настроек и
JSON-файле)
     * @param personalEmail новое значение персональной почты
     * @return результат изменения
     */
    public boolean updateConfigPersonalEmail(String personalEmail) {
        this.personalEmail = personalEmail;
        log.info("Поле personalEmail экземпляра UserConfig обновлено.");
        boolean isUpdateMapAndJSON = this.updateMapAndJSON(PERSONAL_EMAIL_KEY, personalEmail);
        if (!isUpdateMapAndJSON){
            log.warn("Не удалось обновить персональную почту пользователя в конфигурациях: словаре
и/или JSON-файле.");
        }
        return isUpdateMapAndJSON;
    }

    /**
     * Обновляет пару ключ-значение для другой почты в конфигурации (поле, словарь настроек и JSON-

```

```

файле)
    * @param otherEmail новое значение другой почты
    * @return обновленное значение другой почты
    */
    public boolean updateConfigOtherEmail(String otherEmail) {
        this.otherEmail = otherEmail;
        log.info("Поле otherEmail экземпляра UserConfig обновлено.");
        boolean isUpdateMapAndJSON = this.updateMapAndJSON(OTHER_EMAIL_KEY, otherEmail);
        if (!isUpdateMapAndJSON){
            log.warn("Не удалось обновить другую почту пользователя в конфигурациях: словаре и/или
JSON-файле.");
        }
        return isUpdateMapAndJSON;
    }

    @Override
    public String toString() {
        return "UserConfig{packPassword='%s', personalEmail='%s', otherEmail='%s'}"
            .formatted(packPassword, personalEmail, otherEmail);
    }
}

```

Приложение 06. Пакет exception, класс EmailerException

```
package ru.dogobot.Dogobot.exception;

public class EmailerException extends Exception {
    /**
     * Исключение, возникающее при работе с электронной почтой
     * @param message сообщение исключения
     */
    public EmailerException(String message) {
        super(message);
    }
}
```

Приложение 07. Пакет exception, класс FilerException

```
package ru.dogobot.Dogobot.exception;

public class FilerException extends Exception {
    /**
     * Исключение, возникающее при работе с файловой системой.
     * @param message сообщение исключения
     */
    public FilerException(String message) {
        super(message);
    }
}
```

Приложение 08. Пакет model, класс FileDir

```
package ru.dogobot.Dogobot.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Component;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.List;

@Slf4j
@Data
@Component
public class FileDir {

    @Getter
    @AllArgsConstructor
    public enum FDType {
        FILE("Файл"),
        DIR("Папка"),
        OTHER("Неизвестно");

        private final String typeString;
    }

    private File    fdJavaIoFile;
    private String  fdId;
    private FDType  fdType;
    private String  fdNameOriginal;
```

```

private String fdNameInline;
private String fdCallbackData;
private String fdPath;
private Long fdDate;
private Long fdSize;

private File[] fdArray;
private List<List<String>> fdInlineKeyboardIds;

@Override
public String toString() {
    try {
        String separator = System.lineSeparator();
        StringBuilder result = new StringBuilder().append("Информация о папке/файле: ")
.append(separator);
        result.append("Тип: ").append(getFdTypeStringB(separator));
        result.append("Имя: ").append(getFdNameOriginal(separator));
        result.append("Полный путь: ").append(getFdPathStringB(separator));
        result.append("Дата изменения: ").append(getFdDateStringB(separator));
        result.append("Размер: ").append(getFdSizeStringB(separator));
        result.append("Содержимое: ").append(getFdArrayStringB(separator));
        return result.toString();
    } catch (Exception e) {
        String report = "Проблема с файлом/папкой: " + e.getMessage();
        log.error(report);
        return report;
    }
}

/**
 * Собирает строку с содержимым папки/файла
 * @param separator системный разделитель строк
 * @return строка с содержимым папки/файла
 */

```

```

private StringBuilder getFdArrayStringB(String separator) {
    StringBuilder fdArrayStringB = new StringBuilder();

    if (FileDir.this.fdType == FDTType.DIR && FileDir.this.fdArray.length > 2) {
        for (int i = 2; i < FileDir.this.fdArray.length; i++) { //минус родитель и текущий
            fdArrayStringB.append(FileDir.this.fdArray[i].getName()).append(",")
        }
        return fdArrayStringB;
    }

    if (FileDir.this.fdType == FDTType.DIR && FileDir.this.fdArray.length == 2
        || FileDir.this.fdType == FDTType.FILE) {
        fdArrayStringB.append(separator);
        return fdArrayStringB;
    }

    log.error("Проблема с содержимым папки/файла: " + separator
+ Arrays.toString(FileDir.this.fdArray));
    fdArrayStringB.append("Неизвестно, ").append(separator);
    return fdArrayStringB;
}

/**
 * Собирает строку с размером папки/файла
 * @param separator системный разделитель строк
 * @return строка с размером папки/файла
 */
private StringBuilder getFdSizeStringB(String separator) {
    StringBuilder fdSizeStringB = new StringBuilder();
    if (FileDir.this.fdSize != null) {
        fdSizeStringB.append(FileDir.this.fdSize).append(", ").append(separator);
    } else {
        log.error("Проблема с размером папки/файла: " + separator + fdSizeStringB);
    }
}

```



```

        fdSizeStringB.append("Неизвестно,") .append(separator);
    }
    return fdSizeStringB;
}

/**
 * Собирает строку с датой изменения папки/файла
 * @param separator системный разделитель строк
 * @return строка с датой изменения папки/файла
 */
private StringBuilder getFdDateStringB(String separator) {
    StringBuilder fdDateStringB = new StringBuilder();
    if (FileDir.this.fdDate != null) {
        String formattedDate = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss")
        .format(FileDir.this.fdDate);
        fdDateStringB.append("").append(formattedDate).append(",") .append(separator);
    } else {
        log.warn("Дата изменения папки/файла == null " + separator + fdDateStringB);
        fdDateStringB.append("Неизвестно,") .append(separator);
    }
    return fdDateStringB;
}

/**
 * Собирает строку с путём папки/файла
 * @param separator системный разделитель строк
 * @return строка с путём папки/файла
 */
private StringBuilder getFdPathStringB(String separator) {
    StringBuilder fdPathStringB = new StringBuilder();
    if (FileDir.this.fdPath != null) {
        fdPathStringB.append("").append(FileDir.this.fdPath).append(",") .append(separator);
    } else {
        log.error("Путь папки/файла == null " + separator + fdPathStringB);
    }
}

```

```

        fdPathStringB.append("Неизвестно", "").append(separator);
    }
    return fdPathStringB;
}

/**
 * Собирает строку с именем папки/файла
 * @param separator системный разделитель строк
 * @return строка с именем папки/файла
 */
private StringBuilder getFdNameOriginal(String separator) {
    StringBuilder getFdNameOriginalStringB = new StringBuilder();
    if (FileDir.this.fdNameOriginal != null) {
        getFdNameOriginalStringB.append("").append(FileDir.this.fdNameOriginal).append(",")
.append(separator);
    } else {
        log.error("Имя папки/файла == null " + separator + getFdNameOriginalStringB);
        getFdNameOriginalStringB.append("Неизвестно", "").append(separator);
    }
    return getFdNameOriginalStringB;
}

/**
 * Собирает строку с типом папки/файла
 * @param separator системный разделитель строк
 * @return строка с типом папки/файла
 */
private StringBuilder getFdTypeStringB(String separator) {
    StringBuilder fdTypeStringB = new StringBuilder();
    if (FileDir.this.fdType != null) {
        fdTypeStringB.append(FileDir.this.fdType.typeString).append(",")
.append(separator);
    } else {
        log.warn("Тип папки/файла == null " + separator + fdTypeStringB);
        fdTypeStringB.append("Неизвестно", "").append(separator);
    }
}

```

```
    }  
    return fdTypeStringB;  
}  
}
```

Приложение 09. Пакет model, класс User

```
package ru.dogobot.Dogobot.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.*;

import java.sql.Timestamp;

@Data
@Entity
@Table(name = "usersDataTable")
public class User {

    @Id
    @Column(name = "chat_id")
    private Long chatId;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "user_name")
    private String userName;

    @Column(name = "registered_at")
    private Timestamp registeredAt;

    //region Изменяемые параметры пользователя, которые хранятся не только в БД, но в JSON
}
```

```

@Column(name = "pack_password")
private String packPassword;

@Column(name = "personal_email")
private String personalEmail;

@Column(name = "other_email")
private String otherEmail;
//endregion

public User() {
}
public User(
    Long id,
    String firstName,
    String lastName,
    String userName,
    Timestamp timestamp,
    String packPassword,
    String personalEmail,
    String otherEmail
) {
    this.chatId = id;
    this.firstName = firstName;
    this.lastName = lastName;
    this.userName = userName;
    this.registeredAt = timestamp;
    this.packPassword = packPassword;
    this.personalEmail = personalEmail;
    this.otherEmail = otherEmail;
}

@Override
public String toString() {

```

```

String sep = System.lineSeparator();
return "Данные о пользователе." + sep +
    "Id=" + ((chatId == null)? "null" : chatId) + ", " + sep
    + "First Name=" + ((firstName == null)? "null" : firstName) + '\ ' + ", " + sep
    + "Last Name=" + ((lastName == null)? "null" : lastName) + '\ ' + ", " + sep
    + "User Name=" + ((userName == null)? "null" : userName) + '\ ' + ", " + sep
    + "Pack Password=" + ((packPassword == null)? "null" : packPassword) + '\ ' + ", " +
sep
    + "Personal Email=" + ((personalEmail == null)? "null" : personalEmail) + '\ ' + ", "
+ sep
    + "Other Email=" + ((otherEmail == null)? "null" : otherEmail) + '\ ' + ", " + sep
    + "Registered at=" + ((registeredAt == null)? "null" : registeredAt);
    }
}

```

Приложение 10. Пакет model, класс UserRepository

```
package ru.dogobot.Dogobot.model;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
}
```

Приложение 11. Пакет service, класс Archiver

```
package ru.dogobot.Dogobot.service;

import lombok.extern.slf4j.Slf4j;
import net.lingala.zip4j.ZipFile;
import net.lingala.zip4j.model.ZipParameters;
import net.lingala.zip4j.exception.ZipException;
import net.lingala.zip4j.model.enums.CompressionLevel;
import net.lingala.zip4j.model.enums.CompressionMethod;
import net.lingala.zip4j.model.enums.EncryptionMethod;
import org.springframework.stereotype.Service;

import java.io.File;

@Slf4j
@Service
public class Archiver {

    /**
     * Упаковка файла или папки в архив zip без пароля
     *
     * @param sourceForAdd путь к исходному файлу или папке
     * @return путь к сохраненному архиву
     * @throws ZipException если возникли исключения при упаковке
     */
    public String zipFolderWithoutPassword(String sourceForAdd) throws ZipException {
        File fileOrDirForAdd = new File(sourceForAdd);
        if (!fileOrDirForAdd.exists()
            || (!fileOrDirForAdd.isFile() && !fileOrDirForAdd.isDirectory())) {
            throw new ZipException("Файл или папка не найдены и не упакованы (метод без пароля): " +
sourceForAdd);
        }
    }
}
```



```

ZipParameters parameters = new ZipParameters();
parameters.setCompressionMethod(CompressionMethod.DEFLATE);
parameters.setCompressionLevel(CompressionLevel.NORMAL);
parameters.setEncryptFiles(false);
parameters.setEncryptionMethod(EncryptionMethod.NONE);

String zipFilePath = "%s_%s.zip".formatted(
    sourceForAdd,
    java.time.LocalDateTime.now().format(
java.time.format.DateTimeFormatter.ofPattern("HHmmss"))
);
ZipFile zipFile = new ZipFile(zipFilePath);
if (fileOrDirForAdd.isDirectory()) {
    zipFile.addFolder(fileOrDirForAdd, parameters);
    return zipFile.getFile().getAbsolutePath();
}
if (fileOrDirForAdd.isFile()) {
    zipFile.addFile(fileOrDirForAdd, parameters);
    return zipFile.getFile().getAbsolutePath();
}
throw new ZipException("Файл или папка не найдены и не упакованы (метод без пароля): " +
sourceForAdd);
}

/**
 * Упаковка файла или папки в архив zip с паролем
 *
 * @param sourceForAdd путь к исходному файлу или папке
 * @param password     пароль для архива
 * @return путь к сохраненному архиву
 * @throws ZipException если возникли исключения при упаковке
 */
public String zipFolderWithPassword(String sourceForAdd, String password) throws ZipException {

```

```

File fileOrDirForAdd = new File(sourceForAdd);
if (!fileOrDirForAdd.exists()
    || (!fileOrDirForAdd.isFile() && !fileOrDirForAdd.isDirectory()))
{
    throw new ZipException("Файл или папка не найдены и не упакованы (метод без пароля): " +
sourceForAdd);
}

ZipParameters parameters = new ZipParameters();
parameters.setCompressionMethod(CompressionMethod.DEFLATE);
parameters.setCompressionLevel(CompressionLevel.NORMAL);
parameters.setEncryptFiles(false);
parameters.setEncryptionMethod(EncryptionMethod.NONE);

String zipFilePath = "%s_%s.zip".formatted(
    sourceForAdd,
    java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPat-
tern("HHmmss")))
);
ZipFile zipFile = new ZipFile(zipFilePath);
zipFile.setPassword(password.toCharArray());
if (fileOrDirForAdd.isDirectory()) {
    zipFile.addFolder(fileOrDirForAdd, parameters);
    return zipFile.getFile().getAbsolutePath();
}
if (fileOrDirForAdd.isFile()) {
    zipFile.addFile(fileOrDirForAdd, parameters);
    return zipFile.getFile().getAbsolutePath();
}
throw new ZipException("Файл или папка не найдены и не упакованы (метод без пароля): " +
sourceForAdd);
}

/**

```

```

* Распаковка архива zip без пароля
* @param zipFilePath путь к исходному архиву
* @return путь к распакованному архиву
* @throws ZipException если возникли исключения при распаковке
*/
public String unzipFileWithoutPassword(String zipFilePath) throws ZipException {
    if (zipFilePath == null || !new File(zipFilePath).exists()) {
        throw new ZipException("Архив не найден и не распакован (метод без пароля): " + zip-
FilePath);
    }

    ZipFile zipFile = new ZipFile(zipFilePath);
    if (zipFile.isEncrypted()) {
        throw new ZipException("Архив не распакован (метод без пароля), он оказался с паролем.");
    }

    String zipFileParentPath = new File(zipFilePath).getParent();
    zipFile.extractAll(zipFileParentPath);

    return zipFileParentPath;
}

/**
* Распаковка архива zip с паролем
* @param zipFilePath путь к исходному архиву
* @param password пароль для архива
* @return путь к распакованному архиву
* @throws ZipException если возникли исключения при распаковке
*/
public String unzipFileWithPassword(String zipFilePath, String password) throws ZipException {
    if (zipFilePath == null || !new File(zipFilePath).exists()) {
        throw new ZipException("Архив не найден и не распакован (метод без пароля): "
+ zipFilePath);
    }
}

```

```
ZipFile zipFile = new ZipFile(zipFilePath);
if (zipFile.isEncrypted()) {
    zipFile.setPassword(password.toCharArray());
}

String zipFileParentPath = new File(zipFilePath).getParent();
zipFile.extractAll(zipFileParentPath);

return zipFileParentPath;
}
}
```

Приложение 12. Пакет service, класс Emailer

```
package ru.dogobot.Dogobot.service;

import lombok.extern.slf4j.Slf4j;
import org.apache.commons.mail.DefaultAuthenticator;
import org.apache.commons.mail.EmailAttachment;
import org.apache.commons.mail.EmailException;
import org.apache.commons.mail.MultiPartEmail;
import org.springframework.stereotype.Service;
import ru.dogobot.Dogobot.config.EmailConfig;

import javax.mail.*;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Properties;
import java.util.regex.Pattern;

@Slf4j
@Service
public class Emailer {
    final EmailConfig emailConfig;

    public Emailer(EmailConfig emailConfig) {
        this.emailConfig = emailConfig;
    }

    /**
     * Проверяет корректность адреса электронной почты
     * @param email адрес электронной почты для проверки
     * @return true, если адрес корректен, false в противном случае
     */
}
```

```

private boolean isEmailValid(String email) {
    String emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
    return Pattern.compile(emailRegex).matcher(email).matches();
}

/**
 * Отправляет письмо без вложения
 * @param recipient адрес получателя
 * @param subject тема письма
 * @param body текст письма
 * @throws EmailException исключение, возникшее при отправке
 */
public void sendEmailWithoutAttachment(String recipient, String subject, String body) throws
EmailException {
    if (!isEmailValid(recipient)) {
        throw new EmailException("Некорректный адрес электронной почты получателя.");
    }
    String sender = emailConfig.getEmailFrom();
    if (!isEmailValid(sender)) {
        throw new EmailException("Некорректный адрес электронной почты отправителя.");
    }
    String smtpHost = emailConfig.getSmtpHost();
    int smtpPort = emailConfig.getSmtpPort();
    String password = emailConfig.getPassword();
    if (smtpHost == null || smtpPort == 0 || password == null) {
        throw new EmailException("Некорректная конфигурация электронной почты отправителя (SMTP). "
            + System.lineSeparator() + "smtpHost: '" + smtpHost + "', smtpPort: '" + smtpPort
+ "'", также дело может быть в пароле.");
    }

    MultiPartEmail email = new MultiPartEmail();
    email.setHostName(smtpHost);
    email.setSmtpPort(smtpPort);
    email.setAuthenticator(new DefaultAuthenticator(sender, password));
}

```

```

        email.setSSLOnConnect(true);
        email.setFrom(sender);
        email.addTo(recipient);
        email.setSubject(subject);
        email.setMsg(body);
        email.send();
    }

    /**
     * Отправляет письмо с вложением
     * @param recipient адрес получателя
     * @param subject тема письма
     * @param body текст письма
     * @param attachmentPath путь к вложению
     * @return отчёт по отправке
     * @throws EmailException исключение, возникшее при отправке
     */
    public String sendEmailWithAttachment(String recipient, String subject, String body,
String attachmentPath) throws EmailException {
        if (!isEmailValid(recipient)) {
            throw new EmailException("Некорректный адрес электронной почты получателя." + recipient);
        }
        String sender = emailConfig.getEmailFrom();
        if (!isEmailValid(sender)) {
            throw new EmailException("Некорректный адрес электронной почты отправителя." + sender);
        }
        String smtpHost = emailConfig.getSmtpHost();
        int smtpPort = emailConfig.getSmtpPort();
        String password = emailConfig.getPassword();
        if (smtpHost == null || smtpPort == 0 || password == null) {
            throw new EmailException("Некорректная конфигурация электронной почты отправителя (SMTP)."
                + System.lineSeparator() + "smtpHost: '" + smtpHost + "', smtpPort: '" + smtpPort + "',
также дело может быть в пароле.");
        }
    }

```

```

        if (!new File(attachmentPath).exists()) {
            throw new EmailException("Вложение не найдено: " + attachmentPath);
        }

        MultiPartEmail email = new MultiPartEmail();
        email.setHostName(smtpHost);
        email.setSmtpport(smtpPort);
        email.setAuthenticator(new DefaultAuthenticator(sender, password));
        email.setSSLonConnect(true);
        email.setFrom(sender);
        email.addTo(recipient);
        email.setSubject(subject);
        email.setMsg(body);

        EmailAttachment attachment = new EmailAttachment();
        attachment.setPath(attachmentPath);
        attachment.setDisposition(EmailAttachment.ATTACHMENT);
        email.attach(attachment);

        email.send();
        return recipient;
    }

    /**
     * Получает письмо с вложением и сохраняет его в указанную директорию
     * @param dirPathForAttachment путь к директории. ВАЖНО, чтоб в конце был слеш
     * @throws Exception исключения, которые могут возникать при получении
     */
    public void receiveEmailWithAttachment(String dirPathForAttachment) throws Exception {
        //todo метод не используется. Продумать стратегию использования (спец.папка на сервере и про-
        чее)

        // Настройка свойств JavaMail
        Properties props = new Properties();

```



```

почты
    props.setProperty("mail.store.protocol", "imaps"); // Используем протокол IMAP для получения

    props.setProperty("mail.imaps.host", this.emailConfig.getImapHost());

    // Создание сеанса
    Session session = Session.getDefaultInstance(props);

    // Создание хранилища
    Store store = session.getStore("imaps");
    store.connect(this.emailConfig.getImapHost(), this.emailConfig.getEmailFrom(),
this.emailConfig.getPassword());

    // Открытие папки входящих сообщений
    Folder inbox = store.getFolder("INBOX");
    inbox.open(Folder.READ_ONLY);

    // Поиск сообщений с вложениями
    Message[] messages = inbox.getMessage();
    for (Message message : messages) {
        if (message.getContent() instanceof Multipart multipart) {
            for (int i = 0; i < multipart.getCount(); i++) {
                BodyPart bodyPart = multipart.getBodyPart(i);
                if (Part.ATTACHMENT.equalsIgnoreCase(bodyPart.getDisposition())) {
                    // Обработка вложения
                    String fileName = bodyPart.getFileName();
                    // Сохранение вложения
                    InputStream inputStream = bodyPart.getInputStream();
                    OutputStream outputStream = new FileOutputStream(dirPathForAttachment +
fileName);

                    byte[] buffer = new byte[4096];
                    int bytesRead;
                    while ((bytesRead = inputStream.read(buffer)) != -1) {
                        outputStream.write(buffer, 0, bytesRead);
                    }
                }
            }
        }
    }

```

```
        outputStream.close();
        inputStream.close();
    }
}

// Закрытие папки и хранилища
inbox.close(false);
store.close();
}
```

Приложение 13. Пакет service, класс FileManager

```
package ru.dogobot.Dogobot.service;

import com.vdurmont.emoji.EmojiParser;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.meta.api.objects.Update;
import ru.dogobot.Dogobot.exception.EmailerException;
import ru.dogobot.Dogobot.exception.FilerException;
import ru.dogobot.Dogobot.model.FileDir;
import ru.dogobot.Dogobot.model.User;

import java.io.File;
import java.io.FileNotFoundException;
import java.sql.Timestamp;
import java.util.*;
import java.util.stream.Stream;

@Getter
@Slf4j
@Service
public class FileManager {
    final String MENU = "===[ МЕНЮ ]==="; //должно быть менее 32 символов
    final String EXIT_DIR = "[ .. ]"; //должно быть менее 32 символов
    public final static String SELECT_MENU_ITEM = "<< ВЫБЕРИТЕ ПУНКТ МЕНЮ >>"; //должно быть менее 32
СИМВОЛОВ

    @Getter
    @AllArgsConstructor
    public enum FileDirMenu {
```

```

GET_INFO("CBD_FDM_00", "Получить информацию"),
GET_ON_TELEGRAM("CBD_FDM_01", "Получить в Telegram"),
GET_ON_EMAIL("CBD_FDM_10", "Получить на почту"),
SEND_TO_EMAIL("CBD_FDM_11", "Отправить на почту"),
PACK("CBD_FDM_20", "Упаковать в zip"),
PACK_WITH_PASSWORD("CBD_FDM_21", "Упаковать в zip с паролем"),
UNPACK("CBD_FDM_22", "Распаковать из zip"),
UNPACK_WITH_PASSWORD("CBD_FDM_23", "Распаковать из zip с паролем"),
RENAME("CBD_FDM_30", "Переименовать"),
MOVE("CBD_FDM_31", "Переместить"),
COPY("CBD_FDM_32", "Копировать"),
DELETE("CBD_FDM_33", "Удалить"),
TERMINAL("CBD_FDM_40", "Терминал"),
REMOVE_MENU("CBD_FDM_99", "<< УБРАТЬ МЕНЮ >>");
//SEND_TO_TELEGRAM("CBD_FDM_02", "Отправить другу в Telegram");

```

```

private final String buttonCallback;
private final String buttonText;

```

```

}

```

```

@Autowired
Userer userer;

```

```

@Autowired
Screenshoter screenshoter;

```

```

@Autowired
Emailer emailer;

```

```

@Autowired
Archiver archiver;

```

```

@Autowired

```

```

Filer filer;

@Autowired
Terminaler terminaler;

@Autowired
private FileDir fileDir;
private Map<String, FileDir> currentPathDict;

/**
 * Получает полные данные о FileDir (элемент файловой системы)
 * со сканированием содержимого и данных о содержимом.
 * @param inputPath путь к элементу файловой системы
 * @return объект FileDir со всеми данными
 */
public FileDir getFileDirWithScan(String inputPath) {
    try {
        this.fileDir = getFileDirWithoutScan(inputPath);
        return scanFileDirAndSaveItemData(this.fileDir);
    } catch (Exception e) {
        log.error("Полученный путь '" + inputPath + "' некорректен. Следом - открытие домашнего каталога."
                + System.lineSeparator() + e.getMessage());
        return getFileDirHomeWithScan();
    }
}

/**
 * Получает полные данные о FileDir (элемент файловой системы) домашней папки
 * со сканированием содержимого и данных о содержимом.
 * @return объект FileDir со всеми данными
 */
public FileDir getFileDirHomeWithScan() {
    try {

```

```

        this.fileDir = getFileDirWithoutScan(System.getProperty("user.home") + "/forTest"); //todo
убрать forTest
        return scanFileDirAndSaveItemData(this.fileDir);
    } catch (Exception e) {
        log.error("Не открывается даже домашняя папка" + System.lineSeparator() + e.getMessage());
        return null;
    }
}

/**
 * Получает некоторые данные о FileDir (элемент файловой системы)
 * без сканирования содержимого и данных о содержимом.
 * @param inputPath путь к элементу файловой системы
 * @return объект FileDir
 */
protected FileDir getFileDirWithoutScan(String inputPath) {
    FileDir fileDir = new FileDir();
    fileDir.setFdJavaIoFile(getPropertyFdJavaIoFile(inputPath));
    fileDir.setFdId(getPropertyFdId(fileDir));
    fileDir.setFdType(getPropertyFdType(fileDir));
    fileDir.setFdNameOriginal(getPropertyFdNameOriginal(fileDir));
    fileDir.setFdNameInline(getPropertyFdNameInlineButton(fileDir));
    fileDir.setFdCallbackData(getPropertyFdCallbackData(fileDir));
    fileDir.setFdPath(getPropertyFdAbsolutePath(fileDir));
    fileDir.setFdDate(getPropertyFdDate(fileDir));
    fileDir.setFdSize(getPropertyFdLength(fileDir));
    fileDir.setFdArray(getPropertyFdSortArray(fileDir));

    return fileDir;
}

/**
 * Получает java.io.File из переданного на вход пути inputPath
 * @param inputPath путь к элементу файловой системы

```

```

    * @return экземпляр java.io.File
    */
    private File getPropertyFdJavaIoFile(String inputPath) {
        if (inputPath == null
            || inputPath.isBlank()
            || !(new File(inputPath).exists()))
        ) throw new IllegalArgumentException("Путь к элементу файловой системы некорректен.");
        return new File(inputPath);
    }

    /**
     * Получает (создает) идентификатор для FileDir.
     *
     * @param fileDir элемент файловой системы, для которого работает метод
     * @return идентификатор = случайная неповторимая строка на основе текущей даты и части строкового
     * окончания.
     * Длина строки 30 символов, что вписывается в различные требования.
     */
    private String getPropertyFdId(FileDir fileDir) {
        if (fileDir.getFdId() != null) return fileDir.getFdId();
        final String txtForFinish = "%s%s".formatted(
            fileDir.getFdJavaIoFile().getAbsolutePath(), fileDir.getFdJavaIoFile().length()
        );
        return Screenshoter.getRandomStringDate(txtForFinish);
    }

    /**
     * Получает (определяет) тип для FileDir.
     *
     * @param fileDir элемент файловой системы, для которого работает метод
     * @return тип (файл или папка) - enum FileDir.FDType
     */
    private FileDir.FDType getPropertyFdType(FileDir fileDir) {
        if (fileDir.getFdJavaIoFile().isFile())

```

```

        return FileDir.FDType.FILE;
    if (fileDir.getFdJavaIoFile().isDirectory())
        return FileDir.FDType.DIR;
    return FileDir.FDType.OTHER;
}

/**
 * Получает оригинальное название FileDir.
 *
 * @param fileDir элемент файловой системы, для которого работает метод
 * @return оригинальное название
 */
private String getPropertyFdNameOriginal(FileDir fileDir) {
    return fileDir.getFdJavaIoFile().getName();
}

/**
 * Проверяет и в случае необходимости корректирует оригинальное название для соответствия требова-
    ниям к InlineKeyboardButton.
 * Корректировка происходит путём удаления неразрешенных символов и уменьшения длины строки до
    максимально допустимой.
 *
 * @param fileDir элемент файловой системы, для которого работает метод
 * @return проверенное и корректное название для InlineKeyboardButton
 */
protected String getPropertyFdNameInlineButton(FileDir fileDir) {
    //Для поддержки других языков надо либо добавлять сюда,
    //либо менять на стратегию, чтоб в рег.выражении указать только запрещенные символы
    //Регулярное выражение для допустимых символов (^ означает "всё, кроме")
    final String ALLOWED_CHARACTERS_REGEX = "[^a-zA-Za-яA-Я0-9 .,:;`~'\"!@#N$%^&*-
_+=|<>(){}\\[\\]]";
    final int MAX_LENGTH = 30; //еще 2 оставляю для квадратных скобок папок []
    String nameInlineButton = fileDir.getFdNameOriginal();

```



```

        if (this.fileDir.getFdJavaIoFile() != null) {
            if (fileDir.getFdJavaIoFile().equals(this.fileDir.getFdJavaIoFile().getParentFile()))
                return EXIT_DIR;
            if (fileDir.getFdJavaIoFile().equals(this.fileDir.getFdJavaIoFile()))
                return MENU;
        }

        nameInlineButton = nameInlineButton.replaceAll(ALLOWED_CHARACTERS_REGEX, "");
        if (nameInlineButton.length() > MAX_LENGTH) {
            int charactersToRemove = nameInlineButton.length() - MAX_LENGTH + 2;
            int start = nameInlineButton.length() / 2 - charactersToRemove / 2;
            nameInlineButton = nameInlineButton.substring(0, start)
                + ".."
                + nameInlineButton.substring(start + charactersToRemove);
        }
        if (fileDir.getFdType() == FileDir.FDType.DIR) {
            nameInlineButton = "[" + nameInlineButton + "]";
        }

        return nameInlineButton;
    }

    /**
     * Получает (задаёт) команду CallbackData для InlineKeyboardButton.
     *
     * @param fileDir элемент файловой системы, для которого работает метод
     * @return CallbackData команда = FileDir.getFdId()
     */
    private String getPropertyFdCallbackData(FileDir fileDir) {
        return fileDir.getFdId();
    }

    /**
     * Получает полный путь к FileDir.

```

```

*
* @param fileDir элемент файловой системы, для которого работает метод
* @return полный путь
*/
private String getPropertyFdAbsolutePath(FileDir fileDir) {
    return fileDir.getFdJavaIoFile().getAbsolutePath();
}

/**
* Получает дату последнего изменения FileDir.
*
* @param fileDir элемент файловой системы, для которого работает метод
* @return дата последнего изменения
*/
private long getPropertyFdDate(FileDir fileDir) {
    return fileDir.getFdJavaIoFile().lastModified();
}

/**
* Получает занимаемый размер FileDir.
*
* @param fileDir элемент файловой системы, для которого работает метод
* @return занимаемый размер
*/
private long getPropertyFdLength(FileDir fileDir) {
    return fileDir.getFdJavaIoFile().length();
}

/**
* Получает (сортирует и создаёт) список элементов.
* Массив элементов создаётся в следующем порядке:
* - элемент родительской папки;
* - элемент текущей папки;
* - отсортированные элементы внутри папки (папки перед файлами).

```

```

*
* @param fileDir элемент файловой системы, для которого работает метод
* @return отсортированный массив элементов файловой системы, включая текущий и родительский
*/
private File[] getPropertyFdSortArray(FileDir fileDir) {
    File fileByNextPath = fileDir.getFdJavaIoFile();

    //массив по умолчанию
    File[] dirsAndFilesWithDefault = new File[]{
        fileByNextPath,
        fileByNextPath.getParentFile()
    };
    if (fileByNextPath.listFiles() == null)
        return dirsAndFilesWithDefault;

    //сортированный массив элементов ФС внутри папки
    File[] dirsAndFilesWithoutDefault =
        Arrays.stream(fileByNextPath.listFiles())
            .sorted((file1, file2) -> {
                if (file1.isDirectory() && !file2.isDirectory()) {
                    return -1; // Поместить папку перед файлом
                } else if (!file1.isDirectory() && file2.isDirectory()) {
                    return 1; // Поместить файл после папки
                } else {
                    return file1.getName().compareTo(file2.getName()); // Оставить на ме-
сте
                }
            })
            .toArray(File[]::new);

    //копирование - увеличение массива
    return Stream.concat(
        Arrays.stream(dirsAndFilesWithDefault),
        Arrays.stream(dirsAndFilesWithoutDefault)
    );
}

```

```

        )
        .toArray(File[]::new);
    }

    /**
     * Сканирует папку и сохраняет данные о содержимом в FileDir.
     *
     * @param fileDir элемент файловой системы, для которого работает метод
     * @return FileDir, заполненный всеми данными о содержимом
     */
    private FileDir scanFileDirAndSaveItemData(FileDir fileDir) {
        currentPathDict = new HashMap<>();
        List<List<String>> inlineKeyboardIds = new ArrayList<>();

        for (File file : fileDir.getFdArray()) {
            if (file == null)
                continue;
            FileDir fileDirInPath = getFileDirWithoutScan(file.getAbsolutePath());
            currentPathDict.put(fileDirInPath.getFdId(), fileDirInPath);

            List<String> row = new ArrayList<>();
            row.add(currentPathDict.get(fileDirInPath.getFdId()).getFdId());
            inlineKeyboardIds.add(row);
        }

        fileDir.setFdInlineKeyboardIds(
            inlineKeyboardIds
        );

        return fileDir;
    }

    //region РАБОТА С ПОЛЬЗОВАТЕЛЯМИ

```

```

/**
 * Получает ID чата из объекта обновления (как Message, так и CallbackQuery).
 *
 * @param update объект обновления
 * @return ID чата
 */
private Long getChatIdFromUpdate(Update update) {
    Long chatId = null;
    if (update.getMessage() != null) {
        chatId = update.getMessage().getChatId();
    }
    if (update.getCallbackQuery() != null) {
        chatId = update.getCallbackQuery().getFrom().getId();
    }
    return chatId;
}

/**
 * Поиск пользователя в БД или его регистрация в случае отсутствия.
 *
 * @param update объект обновления
 * @return результат поиска
 */
public String findUserOrRegister(Update update) {
    String smileBlush = EmojiParser.parseToUnicode(":blush:");
    String report;
    try {
        User user = userer.findUserById(getChatIdFromUpdate(update));
        report = "Пользователь " + user.getUserName() + " уже зарегистрирован. Вы можете проверить корректность данных, вызвав соответствующую команду из меню" + smileBlush;
        log.info(report);
    } catch (Exception e) {
        try {
            User user = createAndRegisterUser(update);

```

```

        report = "Пользователь " + user.getUserName() + " успешно зарегистрирован!" +
smileBlush;
    } catch (Exception ex) {
        report = "При регистрации произошла ошибка. Пользователь не зарегистрирован или есть
проблемы с БД." + smileBlush;
        log.error(report + System.lineSeparator() + ex.getMessage());
    }
}
return report;
}

/**
 * Создание и регистрация пользователя в БД.
 *
 * @param update объект обновления
 * @return пользователь
 */
public User createAndRegisterUser(Update update) {
    String smileBlush = EmojiParser.parseToUnicode(":blush:");
    String report;
    var message = update.getMessage();

    User user = new User(
        message.getFrom().getId(),
        message.getChat().getFirstName(),
        message.getChat().getLastName(),
        message.getChat().getUserName(),
        new Timestamp(System.currentTimeMillis()),
        userer.getUserConfig().getPackPassword(),
        userer.getUserConfig().getPersonalEmail(),
        userer.getUserConfig().getOtherEmail()
    );

    try {

```

```

        user = userer.registerUser(user);
        report = "Пользователь успешно зарегистрирован!" + smileBlush;
        log.info(report + System.lineSeparator() + user);
    } catch (Exception e) {
        report = "Не удалось зарегистрировать пользователя.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }

    return user;
}

/**
 * Получает информацию о пользователе из БД.
 *
 * @param update объект обновления
 * @return информация о пользователе
 */
public String getUserInfo(Update update) {
    String report;
    User user;
    try {
        String smileBlush = EmojiParser.parseToUnicode(":blush:");
        user = userer.findUserById(getChatIdFromUpdate(update));
        report = "Данные о пользователе найдены " + smileBlush + System.lineSeparator() + user;
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось получить данные о пользователе. Возможно пользователь не зарегистри-
рован или есть проблемы с БД.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }

    return report;
}

```

```

/**
 * Удаляет пользователя из БД
 *
 * @param update объект обновления
 * @return пользователь
 */
public String deleteUser(Update update) {
    String report;
    try {
        userer.deleteUser(getChatIdFromUpdate(update));
        report = "Данные о пользователе удалены из БД.";
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось удалить данные о пользователе. Возможно пользователь не зарегистриро-
ван или есть проблемы с БД.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Обновляет пароль упаковки/распаковки
 *
 * @param update объект обновления
 * @param newPackPassword новый пароль
 * @return пользователь с обновленным паролем
 */
public String updatePackPassword(Update update, String newPackPassword) {
    String report;
    User user;
    try {
        user = userer.updatePackPassword(getChatIdFromUpdate(update), newPackPassword);
        report = "Пароль упаковки/распаковки успешно обновлен." + System.lineSeparator()
+ user.getPackPassword();
    }
}

```



```

        log.info(report);
    } catch (Exception e) {
        report = "При попытке обновления пароля упаковки/распаковки для пользователя возникло ис-
ключение.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Обновляет персональный адрес электронной почты
 *
 * @param update      объект обновления
 * @param newPersonalMail новый персональный адрес
 * @return пользователь с обновленным персональным адресом
 */
public String updatePersonalMail(Update update, String newPersonalMail) {
    String report;
    User user;
    try {
        user = userer.updatePersonalEmail(getChatIdFromUpdate(update), newPersonalMail);
        report = "Персональный адрес электронной почты успешно обновлен." + System.lineSeparator()
+ user.getPersonalEmail();
        log.info(report);
    } catch (Exception e) {
        report = "При попытке обновления персонального адреса электронной почты для пользователя
возникло исключение.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Обновляет другой адрес электронной почты

```

```

*
* @param update      объект обновления
* @param newOtherMail новый другой адрес
* @return пользователь с обновленным другой адрес
*/
public String updateOtherMail(Update update, String newOtherMail) {
    String report;
    User user;
    try {
        user = userer.updateOtherEmail(getChatIdFromUpdate(update), newOtherMail);
        report = "Другой адрес электронной почты успешно обновлен." + System.lineSeparator()
+ user.getOtherEmail();
        log.info(report);
    } catch (Exception e) {
        report = "При попытке обновления другого адреса электронной почты для пользователя воз-
никло исключение.";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
* Получает личные настройки пользователя
*
* @param update объект обновления
* @return строку с личными настройками
*/
public String getUserSettings(Update update) {
    String report;
    String sep = System.lineSeparator();
    try {
        User user = userer.findUserById(getChatIdFromUpdate(update));
        report = "Личные настройки:" + sep +
            "---" + sep +

```

```

        "Пароль (упаковка, распаковка и т.п.): " + user.getPackPassword() + sep +
        "Личная почта (для получения на неё писем): " + user.getPersonalEmail() + sep +
        "Другая почта (для отправки на неё писем): " + user.getOtherEmail() + sep;
    log.info("Личные настройки пользователя получены." + System.lineSeparator());
} catch (Exception e) {
    report = "Не удалось получить личные настройки.";
    log.error(report + sep + e.getMessage());
}
return report;
}

//endregion

/**
 * Делает скриншот
 *
 * @return путь к скриншоту
 */
public String printScreen() {
    try {
        String screenPath = getScreenshot().take();
        log.info("Скриншот сделан. Путь: " + screenPath);
        return screenPath;
    } catch (Exception e) {
        log.warn("Не удалось сделать скриншот. " + e.getMessage());
        return null;
    }
}

//region РАБОТА С ЭЛЕКТРОННОЙ ПОЧТОЙ

/**
 * Отправляет письмо по электронной почте с вложением fileDir
 *

```

```

* @param fileDir    элемент файловой системы, для отправки
* @param recipient  адрес получателя
* @return отчёт отправки
*/
protected String sendEmailWithAttachment(FileDir fileDir, String recipient) {
    String report;
    try {
        if (!fileDir.getFdJavaIoFile().exists()) {
            throw new FileNotFoundException();
        }

        if (!fileDir.getFdType().equals(FileDir.FDType.FILE)) {
            throw new EmailerException("Элемент файловой системы не является файлом. Если это папка, то перед отправкой, её стоит упаковать в архив.");
        }

        String pathToAttachment = fileDir.getFdPath();
        String subject = (pathToAttachment.length() > 30) ?
            "....%s".formatted(pathToAttachment.substring(pathToAttachment.length() - 30))
            : pathToAttachment;
        String text = fileDir.toString();
        report = "Отправка письма с вложением прошла без исключений." + System.lineSeparator()
            + "Получатель: " + emailer.sendEmailWithAttachment(recipient, subject, text,
pathToAttachment);
        log.info(report);
    } catch (FileNotFoundException e) {
        report = "Не удалось отправить письмо с вложением. Файл не существует. ";
        log.error(report + System.lineSeparator() + e.getMessage());
    } catch (EmailerException e) {
        report = "Не удалось отправить письмо с вложением. " + System.lineSeparator() +
e.getMessage();
        log.error(report);
    } catch (Exception e) {
        report = "Не удалось отправить письмо с вложением. Проверьте корректность параметров.";
    }
}

```

```

        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Отправляет письмо с вложением на личную почту
 *
 * @param fileDir элемент файловой системы, для отправки
 * @param update объект обновления
 * @return отчёт отправки
 */
public String sendEmailPersonal(FileDir fileDir, Update update) {
    String report;
    try {
        String recipient = userer.findUserById(getChatIdFromUpdate(update)).getPersonalEmail();
        report = sendEmailWithAttachment(fileDir, recipient);
    } catch (Exception e) {
        report = "Не удалось отправить письмо с вложением (или получить адрес личной почты). ";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Отправляет письмо с вложением на другую почту
 *
 * @param fileDir элемент файловой системы, для отправки
 * @param update объект обновления
 * @return отчёт отправки
 */
public String sendEmailOther(FileDir fileDir, Update update) {
    String report;
    try {

```

```

        String recipient = userer.findUserById(getChatIdFromUpdate(update)).getOtherEmail();
        report = sendEmailWithAttachment(fileDir, recipient);
    } catch (Exception e) {
        report = "Не удалось отправить письмо с вложением (или получить адрес другой почты). ";
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

//endregion

//region РАБОТА С АРХИВАМИ

/**
 * Упаковывает файл или папку (метод без пароля)
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод
 * @return отчет об упаковке
 */
public String zipFileDirWithoutPassword(FileDir sourceFileDir) {
    String report;
    try {
        report = "Упаковка файла или папки (метод без пароля) прошла без исключений. " +
System.lineSeparator()
            + archiver.zipFolderWithoutPassword(sourceFileDir.getFdPath());
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось упаковать файл или папку (метод без пароля): " +
System.lineSeparator()
            + sourceFileDir.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

```

```

/**
 * Упаковывает файл или папку (метод с паролем).
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод
 * @param password      пароль для упаковки
 * @return отчет об упаковке
 */
public String zipFileDirWithPassword(FileDir sourceFileDir, String password) {
    String report;
    try {
        report = "Упаковка файла или папки (метод с паролем) прошла без исключений. " +
System.lineSeparator()
                + archiver.zipFolderWithPassword(sourceFileDir.getFdPath(), password);
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось упаковать файл или папку (метод без пароля): " +
System.lineSeparator()
                + sourceFileDir.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Упаковывает файл или папку (метод с паролем - перегрузка без указания пароля).
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод
 * @param update        объект обновления
 * @return отчет об упаковке
 */
public String zipFileDirWithPassword(FileDir sourceFileDir, Update update) {
    String report;
    try {

```

```

        String password = userer.findUserById(getChatIdFromUpdate(update)).getPackPassword();
        report = zipFileDirWithPassword(sourceFileDir, password);
    } catch (Exception e) {
        report = "Не удалось получить пароль или не удалось упаковать файл/папку (метод с паролем
- перегрузка без указания пароля): " + System.lineSeparator()
            + sourceFileDir.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Распаковывает файл или папку (метод без пароля).
 * Если в родительской папке архива есть файлы или папки с именами, как в архиве, то они будут за-
менены.
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод.
 * @return отчет о распаковке
 */
public String unzipFileDirWithoutPassword(FileDir sourceFileDir) {
    String report;
    try {
        report = "Распаковка файла или папки (метод без пароля) прошла без исключений. " +
System.lineSeparator()
            + archiver.unzipFileWithoutPassword(sourceFileDir.getFdPath()) +
System.lineSeparator()
            + "Обратите внимание, если в родительской папке архива '" +
sourceFileDir.getFdPath() + "' были файлы или папки с именами, как в распакованном архиве, то они были
заменены на распакованные.";
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось распаковать файл или папку (метод без пароля): " +
System.lineSeparator()
            + sourceFileDir.getFdPath();
    }
}

```



```

        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Распаковывает файл или папку (метод с паролем).
 * Если в родительской папке архива есть файлы или папки с именами, как в архиве, то они будут за-
менены.
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод
 * @param password       пароль для распаковки
 * @return отчет о распаковке
 */
public String unzipFileDirWithPassword(FileDir sourceFileDir, String password) {
    String report;
    try {
        report = "Распаковка файла или папки (метод с паролем) прошла без исключений. " +
System.lineSeparator()
                + archiver.unzipFileWithPassword(sourceFileDir.getFdPath(), password) +
System.lineSeparator()
                + "Обратите внимание, если в родительской папке архива '" +
sourceFileDir.getFdPath() + "' были файлы или папки с именами, как в распакованном архиве, то они были
заменены на распакованные.";
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось распаковать файл или папку (метод с паролем): " +
System.lineSeparator()
                + sourceFileDir.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

```

```

/**
 * Распаковывает файл или папку (метод с паролем - перегрузка без указания пароля).
 * Если в родительской папке архива есть файлы или папки с именами, как в архиве, то они будут за-
менены.
 *
 * @param sourceFileDir элемент файловой системы, для которого работает метод
 * @param update        объект обновления
 * @return отчет о распаковке
 */
public String unzipFileDirWithPassword(FileDir sourceFileDir, Update update) {
    String report;
    try {
        String password = userer.findUserById(getChatIdFromUpdate(update)).getPackPassword();
        report = unzipFileDirWithPassword(sourceFileDir, password);
    } catch (Exception e) {
        report = "Не удалось получить пароль или не удалось распаковать файл/папку (метод с паро-
лем - перегрузка без указания пароля): " + System.lineSeparator()
            + sourceFileDir.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

//endregion

//region РАБОТА С ФАЙЛАМИ

/**
 * Переименовывает папку или файл
 *
 * @param oldFileDir элемент файловой системы для которого работает метод
 * @param newName     новое имя
 * @return отчет о переименовании
 */

```

```

public String fileDirRename(FileDir oldFileDir, String newName) {
    String report;
    try {
        File newFile = filer.renameFileDir(oldFileDir.getFdJavaIoFile(), newName);
        report = "Папка или файл: " + oldFileDir.getFdNameOriginal() + " переименован в: " +
newFile.getName();
        log.info(report);
    } catch (FilerException e) {
        report = "Не удалось переименовать папку или файл. Подробности:" + System.lineSeparator()
+ e.getMessage();
        log.error(report);
    } catch (Exception e) {
        report = "Не удалось переименовать папку или файл.";
        log.error(report + " Подробности:" + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Перемещает папку или файл
 *
 * @param oldFileDir элемент файловой системы для которого работает метод
 * @param newPathParent путь к родительской папке нового места
 * @return отчет о перемещении
 */
public String fileDirMove(FileDir oldFileDir, String newPathParent) {
    String report;
    try {
        File newFile = filer.moveFileDir(
            oldFileDir.getFdJavaIoFile(),
            new File(newPathParent, oldFileDir.getFdNameOriginal()).getAbsolutePath()
        );
        report = "Папка или файл: " + oldFileDir.getFdPath() + " перемещен в: " +
newFile.getAbsolutePath();
    }
}

```

```

        log.info(report);
    } catch (FilerException e) {
        report = "Не удалось переместить папку или файл. Подробности:" + System.lineSeparator() +
e.getMessage();
        log.error(report);
    } catch (Exception e) {
        report = "Не удалось переименовать папку или файл.";
        log.error(report + " Подробности:" + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Копирует папку или файл
 *
 * @param fileDire      элемент файловой системы для которого работает метод
 * @param newPathParent путь к родительской папке нового места
 * @return отчет о копировании
 */
public String fileDirCopy(FileDir fileDire, String newPathParent) {
    String report;
    File newFile;
    try {
        newFile = filer.copyFileDir(
            fileDire.getFdJavaIoFile().toPath(),
            (new File(newPathParent, fileDire.getFdNameOriginal())).toPath()
        );
        report = "Папка или файл: " + fileDire.getFdPath() + " скопирована в: " + newPathParent;
        log.info(report);
    } catch (FilerException e) {
        report = "Копирование аварийно остановлено. Причина - не удалось скопировать папку или
файл. Подробности:" + System.lineSeparator() + e.getMessage();
        log.error(report);
    } catch (Exception e) {

```

```

        report = "Копирование аварийно остановлено. Причина - не удалось скопировать папку или
файл: " + fileDire.getFdPath();
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

/**
 * Удаляет папку или файл
 *
 * @param fileDir элемент файловой системы для которого работает метод
 * @return отчет об удалении
 */
public String fileDirDelete(FileDir fileDir) {
    String report;
    try {
        filer.deleteFileDir(fileDir.getFdJavaIoFile());
        report = "Папка или файл: " + fileDir.getFdPath() + " удалена";
    } catch (Exception e) {
        report = "Не удалось удалить папку или файл: " + fileDir.getFdPath() + e.getMessage();
        log.error(report);
    }
    return report;
}

//endregion

/**
 * Запускает команду(ы) в терминале
 *
 * @param script команда(ы)
 * @return отчет о выполнении
 */
public String terminalExecute(String script) {

```

```

String report;
try {
    report = terminaler.processBuilderExecuteWithAnswer(script);
    log.info(report);
} catch (Exception e) {
    report = "Не удалось выполнить команду(ы) в терминале: " + script;
    log.error(report + System.lineSeparator() + e.getMessage());
}
return report;
}

/**
 * Отложенный запуск копии бота и завершение текущей
 * @return отчет о выполнении
 */
public String botReset() {
    String report;
    try {
        String javaBin = System.getProperty("java.home") + File.separator + "bin" + File.separator
+ "java";
        String currentJar = new File(new File("."), "Dogobot-0.0.1-
SNAPSHOT.jar").getAbsolutePath();
        String sleep = "sleep 5";
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
            sleep = "timeout /t 5 > nul";
        }
        terminaler.processBuilderExecute(sleep + " && " + javaBin + " -jar " + currentJar);
        report = "Отложенный запуск копии бота запущен. Следом запись вот этих логов в файл, за-
вершение программы. Иначе - перехват исключений.";
        log.info(report);
        System.exit(0);
    } catch (Exception e) {
        report = "Не удалось отложено запустить копию бота. Выключаю.";
        log.error(report + " Подробности:" + System.lineSeparator() + e.getMessage());
    }
}

```

```
        System.exit(0);  
    }  
    return report;  
}  
  
}
```

Приложение 14. Пакет service, класс Filer

```
package ru.dogobot.Dogobot.service;

import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import ru.dogobot.Dogobot.exception.FilerException;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;

@Slf4j
@Service
public class Filer {

    /**
     * Переименовывает папку или файл
     * @param source исходный файл или папка для переименования
     * @param newName новое имя
     * @return новый файл или папка с новым именем
     * @throws FilerException если возникли исключения
     */
    public File renameFileDir(File source, String newName) throws FilerException {
        if (!source.exists()) {
            throw new FilerException("Ссылка на файл или папку для переименования не корректна: '" +
source.getName() + "'");
        }
        if (newName.isEmpty()) {
            throw new FilerException("Новое имя пустое для: '" + source.getName() + "'");
        }
        if (source.getName().equals(newName)) {
```



```

        throw new FilerException("Старое и новое имена совпадают: '" + source.getName() + "'");
    }

    File fileWithNewName = new File(source.getParent(), newName);
    if (!source.renameTo(fileWithNewName)) {
        throw new FilerException("Не удалось переименовать папку или файл: '" + source.getName() +
"' в: '" + newName + "'");
    }
    return fileWithNewName;
}

/**
 * Перемещает папку или файл
 * @param source исходный файл или папка для перемещения
 * @param destinationPath адрес к новому пути
 * @return новый файл или папка с новым путем
 * @throws FilerException если возникли исключения
 */
public File moveFileDir(File source, String destinationPath) throws FilerException {
    if (!source.exists()) {
        throw new FilerException("Ссылка на файл или папку для перемещения не корректна: '" +
source.getName() + "'");
    }
    if (destinationPath.isEmpty()) {
        throw new FilerException("Адрес к новому пути пустой для: '" + source.getName() + "'");
    }
    if (source.getName().equals(destinationPath)) {
        throw new FilerException("Старый и новый пути совпадают: '" + source.getAbsolutePath() +
"'");
    }

    File fileWithNewPath = new File(destinationPath);
    if (!source.renameTo(fileWithNewPath)) {
        throw new FilerException("Не удалось переместить папку или файл: '" +

```

```

source.getAbsolutePath() + "' в: '" + destinationPath + "'");
    }
    return fileWithNewPath;
}

/**
 * Копирует папку или файл
 * @param source путь к исходному файлу или папке
 * @param destination путь к новому месту
 * @return новый файл или папка
 * @throws IOException если возникли исключения
 * @throws FileNotFoundException если возникли исключения
 */
public File copyFileDir(Path source, Path destination) throws IOException, FileNotFoundException {
    if (source == null || !source.toFile().exists() || destination == null) {
        throw new FileNotFoundException("Ссылки на исходный файл/папку или папку для копирования не корректны");
    }

    if (!Files.isDirectory(source)) {
        return Files.copy(source, destination, StandardCopyOption.REPLACE_EXISTING)
            .toFile();
    }

    if (!Files.exists(destination)) {
        Files.createDirectories(destination);
    }

    Files.walk(source)
        .forEach(sourcePath -> {
            Path destPath = destination.resolve(source.relativeTo(sourcePath));
            try {
                if (Files.isDirectory(sourcePath)) {
                    if (!Files.exists(destPath)) {
                        Files.createDirectory(destPath);
                    }
                }
            }
        });
}

```

```

        }
        } else {
            Files.copy(sourcePath, destPath, StandardCopyOption.REPLACE_EXISTING);
        }
    } catch (IOException e) {
        throw new RuntimeException("Копирование аварийно остановлено. Причина - не
удалось скопировать '" + sourcePath + "' в '" + destPath + "' " + System.lineSeparator() + e.getMes-
sage());
    }
    });
    return destination.toFile();
}

/**
 * Рекурсивно удаляет папку или файл
 * @param fileOrDir исходный файл или папка
 */
public void deleteFileDir(File fileOrDir) {
    // Recursive deletion for directories
    if (fileOrDir.isDirectory()) {
        File[] contents = fileOrDir.listFiles();
        if (contents != null) {
            for (File f : contents) {
                deleteFileDir(f);
            }
        }
    }
    // Delete the file or empty directory
    fileOrDir.delete();
}
}

```

Приложение 15. Пакет service, класс Jsoner

```
package ru.dogobot.Dogobot.service;

import lombok.Getter;
import lombok.Setter;
import lombok.extern.slf4j.Slf4j;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONTokener;
import org.springframework.stereotype.Service;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

@Slf4j
@Setter
@Getter
@Service
public class Jsoner {

    /**
     * Метод, получающий значение по ключу из JSON-файла
     * @param filePath путь к файлу JSON,
     * @param jsonKey ключ, по которому необходимо найти значение
     * @return искомое значение
     */
    public String readJSONFile(String filePath, String jsonKey) {
        String jsonValue = null;
        try (FileReader reader = new FileReader(filePath))
        {
            JSONTokener tokenener = new JSONTokener(reader);
```

```

        JSONObject jsonObject = new JSONObject(tokener);
        jsonValue = jsonObject.getString(jsonKey);
    } catch (FileNotFoundException e) {
        log.warn("Файл не найден: " + filePath + System.lineSeparator() + e.getMessage());
    } catch (JSONException e) {
        log.warn("Ошибка при распознавании JSON. Ключ: " + jsonKey + System.lineSeparator() +
e.getMessage());
    } catch (IOException e) {
        log.warn("Проблема с вводом-выводом при чтении файла: " + filePath + System.lineSepara-
tor() + e.getMessage());
    }
    return jsonValue;
}

/**
 * Изменяет (добавляет, в случае отсутствия) значение по ключу в JSON-файле.
 * @param jsonKey ключ, по которому необходимо изменить значение
 * @param jsonValue новое значение
 * @return результат изменения
 */
public boolean updateValueJSONFile(String filePath, int indentFactor, String jsonKey, String json-
Value) {
    boolean result = false;
    try {
        JSONObject jsonObject;

        try (FileReader reader = new FileReader(filePath)) {
            JSOMTokener tokener = new JSOMTokener(reader);
            jsonObject = new JSONObject(tokener);
            jsonObject.put(jsonKey, jsonValue);
        }

        try (FileWriter writer = new FileWriter(filePath)) {
            writer.write(jsonObject.toString(indentFactor));
        }
    }

```

```

    }

    result = true;
} catch (FileNotFoundException e) {
    log.error("Файл не найден: " + filePath + System.lineSeparator() + e.getMessage());
} catch (JSONException e) {
    log.error("Ошибка при распознавании JSON. Ключ: " + jsonKey + System.lineSeparator() +
e.getMessage());
} catch (IOException e) {
    log.error("Проблема с вводом-выводом при изменении файла: " + filePath + System.lineSepa-
rator() + e.getMessage());
}

return result;
}
}

```

Приложение 16. Пакет service, класс Screenshoter

```
package ru.dogobot.Dogobot.service;

import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

@Slf4j
@Service
public class Screenshoter {

    /**
     * Делает скриншот экрана
     * @return полный путь к скриншоту
     */
    public String take() {
        System.setProperty("java.awt.headless", "false");
        GraphicsEnvironment ge;
        GraphicsDevice[] screens = new GraphicsDevice[0];
        try {
            ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
            screens = ge.getScreenDevices();
        } catch (HeadlessException e) {
            log.error("Не удалось получить список экранов для скриншота: " + e.getMessage());
        }

        Rectangle allScreenBounds = new Rectangle();
```

```

for (GraphicsDevice screen : screens) {
    Rectangle screenBounds = screen.getDefaultConfiguration().getBounds();
    allScreenBounds.width += screenBounds.width;
    allScreenBounds.height = Math.max(allScreenBounds.height, screenBounds.height);
}

BufferedImage capture = null;
try {
    capture = new Robot().createScreenCapture(allScreenBounds);
} catch (AWTException e) {
    log.error("Не удалось захватить изображение для скриншота: " + e.getMessage());
}

String screenshotDirPath = "../screenshots";
String screenName = getRandomStringDate(String.valueOf(new Random().nextInt(10)));
String screenExtension = "png";
String screenPath = String.format("%s/%s.%s", screenshotDirPath, screenName, screenExtension);

File dir = new File(screenshotDirPath);
if (!dir.exists() && !dir.mkdirs()) {
    log.error("Не удалось создать каталог для скриншотов");
}

File imageFile = null;
try {
    imageFile = new File(screenPath);
} catch (Exception e) {
    log.error("Не удалось создать файл для скриншота: " + e.getMessage());
}

try {
    ImageIO.write(capture, screenExtension, imageFile);
} catch (IOException e) {

```



```

        log.error("Не удалось сохранить изображение для скриншота: " + e.getMessage());
    }

    return screenPath;
}

/**
 * Генерирует случайную неповторимую строку на основе текущей даты и части строкового окончания.
 * Длина строки 30 символов, что вписывается в различные требования.
 * @param txtForFinish текст для окончания (исходные данные, из них возьмет 6 предпоследних симво-
лов)
 * @return случайная неповторимая строка
 */
protected static String getRandomStringDate(String txtForFinish) {
    String dateTimeNow = java.time.LocalDateTime.now().format(
        java.time.format.DateTimeFormatter.ofPattern("yyMMdd_HHmmss_SSSSSSSSS")
    );
    if (txtForFinish.length() > 6)
        txtForFinish = txtForFinish.substring(txtForFinish.length() - 7, txtForFinish.length() -
1);
    return String.format("%s_%s",
        dateTimeNow, txtForFinish
    );
}
}

```

Приложение 17. Пакет service, класс Terminaler

```
package ru.dogobot.Dogobot.service;

import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.io.*;

@Slf4j
@Service
public class Terminaler {

    /**
     * Запускает команду(ы) в терминале
     * @param script команда(ы) для запуска в терминале
     * @return процесс запуска команды
     * @throws IOException если возникли исключения
     */
    public Process processBuilderExecute(String script) throws IOException {
        String[] bashOrCmdAndScript = fillCommandForOS(script);
        ProcessBuilder builderForScript = new ProcessBuilder(bashOrCmdAndScript);
        return builderForScript.start();
    }

    /**
     * Заполняет команду для запуска в терминале с учетом ОС
     * @param script команда для запуска
     * @return команду для запуска в необходимом формате (String[])
     */
    private String[] fillCommandForOS(String script) {
        String bashOrCmd1 = "bash";
        String bashOrCmd2 = "-c";
        if (System.getProperty("os.name").toLowerCase().contains("win")) {
```

```

        bashOrCmd1 = "cmd";
        bashOrCmd2 = "/c";
    }
    script = script.replace("-", "--");
    return new String[]{bashOrCmd1, bashOrCmd2, script};
}

private StringBuilder getAnswerFromProcessStream(Process processForScript) throws IOException {
    StringBuilder report = new StringBuilder();
    InputStream is = processForScript.getInputStream();
    BufferedReader reader = new BufferedReader(new InputStreamReader(is));
    String line;
    while ((line = reader.readLine()) != null) {
        report.append(line).append("\n");
    }
    return report;
}

/**
 * Запускает команду(ы) в терминале и возвращает ответ
 * @param script команда(ы) для запуска в терминале
 * @return ответ от команды в полученном процессе
 * @throws IOException если возникли исключения
 */
public String processBuilderExecuteWithAnswer(String script) throws IOException {
    Process processForScript = processBuilderExecute(script);
    return getAnswerFromProcessStream(processForScript).toString();
}

/**
 * Запускает команду(ы) в терминале и возвращает ответ - второй запасной способ
 * @param script команда(ы) для запуска в терминале
 * @return ответ команды

```

```
    * @throws IOException если возникли исключения
    */
    public String processExecuteWithAnswer(String script) throws IOException {
        String[] bashOrCmdAndScript = fillCommandForOS(script);
        Runtime rt = Runtime.getRuntime();
        Process processForScript = rt.exec(bashOrCmdAndScript);
        return getAnswerFromProcessStream(processForScript).toString();
    }
}
```

Приложение 18. Пакет service, класс Userer

```
package ru.dogobot.Dogobot.service;

import lombok.Data;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.dogobot.Dogobot.config.UserConfig;
import ru.dogobot.Dogobot.model.User;
import ru.dogobot.Dogobot.model.UserRepository;

import java.util.NoSuchElementException;

@Slf4j
@Getter
@Service
@Data
public class Userer {

    @Autowired
    private UserRepository userRepository;

    private UserConfig userConfig;

    public Userer(UserConfig userConfig) {
        this.userConfig = userConfig;
    }

    /**
     * Ищет пользователя в базе данных по chatId
     * @param chatId Id чата
     * @return найденный пользователь
     */
}
```

```

    * @throws NoSuchElementException если пользователь не найден
    */
    protected User findUserById(Long chatId) throws NoSuchElementException {
        return userRepository.findById(chatId).get();
    }

    /**
     * Регистрирует пользователя в базе данных
     * @param userForRegister пользователь
     * @return пользователь для регистрации
     */
    protected User registerUser(User userForRegister) {
        return userRepository.save(userForRegister);
    }

    /**
     * Удаляет пользователя из базы данных
     * @param chatId Id чата
     */
    protected void deleteUser(Long chatId) {
        User user = findUserById(chatId);
        userRepository.delete(user);
        user = null;
    }

    /**
     * Обновляет пароль пользователя в базе данных
     * @param chatId Id чата
     * @param newPackPassword новый пароль
     * @return пользователь
     */
    protected User updatePackPassword(Long chatId, String newPackPassword) {
        User user = findUserById(chatId);
        //в конфигурациях
    }

```

```

        if (!userConfig.updateConfigPackPassword(newPackPassword)) {
            return user;
        }
        //в экземпляре user
        user.setPackPassword(userConfig.getConfigs().get(userConfig.getPACK_PASSWORD_KEY()));
        //в БД
        userRepository.save(user);
        log.info("Пароль пользователя обновлен, как локально, так и в БД.");
        return user;
    }

    /**
     * Обновляет персональный адрес электронной почты пользователя
     * @param chatId Id чата
     * @param newPersonalMail новый персональный адрес электронной почты
     * @return пользователь
     */
    protected User updatePersonalEmail(Long chatId, String newPersonalMail) {
        User user = findUserById(chatId);
        //в конфигурациях
        if (!userConfig.updateConfigPersonalEmail(newPersonalMail)) {
            return user;
        }
        //в экземпляре user
        user.setPersonalEmail(userConfig.getConfigs().get(userConfig.getPERSONAL_EMAIL_KEY()));
        //в БД
        userRepository.save(user);
        log.info("Персональный адрес пользователя обновлен, как локально, так и в БД.");
        return user;
    }

    /**
     * Обновляет другой адрес электронной почты пользователя
     * @param chatId Id чата

```

```

    * @param newOtherMail новый другой адрес электронной почты
    * @return пользователь
    */
    protected User updateOtherEmail(Long chatId, String newOtherMail) {
        User user = findUserById(chatId);
        //в конфигурациях
        if (!userConfig.updateConfigOtherEmail(newOtherMail)) {
            return user;
        }
        //в экземпляре user
        user.setOtherEmail(userConfig.getConfigs().get(userConfig.getOTHER_EMAIL_KEY()));
        //в БД
        userRepository.save(user);
        log.info("Другой другой адрес электронной почты пользователя обновлен, как локально, так и в
БД.");
        return user;
    }
}

```


Приложение 19. Пакет view_controller, класс TelegramBot

```
package ru.dogobot.Dogobot.view_controller;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.telegram.telegrambots.bots.TelegramLongPollingBot;
import org.telegram.telegrambots.meta.api.methods.commands.SetMyCommands;
import org.telegram.telegrambots.meta.api.methods.send.SendDocument;
import org.telegram.telegrambots.meta.api.methods.send.SendMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.DeleteMessage;
import org.telegram.telegrambots.meta.api.methods.updatingmessages.EditMessageText;
import org.telegram.telegrambots.meta.api.objects.InputFile;
import org.telegram.telegrambots.meta.api.objects.Update;
import org.telegram.telegrambots.meta.api.objects.commands.BotCommand;
import org.telegram.telegrambots.meta.api.objects.commands.scope.BotCommandScopeDefault;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.InlineKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.ReplyKeyboardMarkup;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.InlineKeyboardButton;
import org.telegram.telegrambots.meta.api.objects.replykeyboard.buttons.KeyboardRow;
import org.telegram.telegrambots.meta.exceptions.TelegramApiException;
import ru.dogobot.Dogobot.config.BotConfig;
import ru.dogobot.Dogobot.model.FileDir;
import ru.dogobot.Dogobot.service.FileManager;

import java.io.*;
import java.util.*;

@Slf4j
@Service
public class TelegramBot extends TelegramLongPollingBot {
```

```

//todo Посмотреть модификаторы доступа

//region КОНСТАНТЫ и другие исходные данные
final BotConfig botConfig;

//команды меню
@Getter
@AllArgsConstructor
public enum BotMenuEnum {
    START("/start", "Здравствуйте!"),
    SHOWHOME("/showhome", "Показать домашнюю папку"),
    GOTO("/goto", "Перейти по адресу"),
    SCREENSHOT("/screenshot", "Сделать скриншот"),
    MYDATA("/mydata", "Посмотреть данные о себе"),
    DELETEDATA("/deletedata", "Удалить данные о себе"),
    HELP("/help", "Помощь"),
    SETTINGS("/settings", "Настройки"),
    RESET("/botreset", "Перезапустить бота"),
    EXIT("/botstop", "Остановить бота");

    private final String key;
    private final String description;
}

final List<BotCommand> botMenu;

//нижняя клавиатура по умолчанию (только наименования в таблице)
@Getter
@AllArgsConstructor
public enum ReplyKeyboardEnum {
    ROW1_BUTTON1("Домой", "Показать домашнюю папку"),
    ROW1_BUTTON2("Настройки", "Настройки");
}

```

```

        private final String key;
        private final String description;
    }

    List<List<String>> replyKeyboardNames = new ArrayList<>()
        List.of(Arrays.asList(ReplyKeyboardEnum.ROW1_BUTTON1.key,
ReplyKeyboardEnum.ROW1_BUTTON2.key))
    );

    //другие команды через дробь
    @Getter
    @AllArgsConstructor
    public enum OtherCommandEnum {
        GOTO("/goto ", "Перейти в указанную папку"),
        SETPASS("/setpass ", "Установить новый пароль"),
        SETPMAIL("/setpmail ", "Установить новый адрес 'персональной электронной почты'"),
        SETOMAIL("/setomail ", "Установить новый адрес 'другой электронной почты'"),
        RENAME("/rn ", "Переименовать папку/файл"),
        MOVE("/mv ", "Переместить папку/файл"),
        COPY("/cp ", "Копировать папку/файл"),
        CMD("/cmd ", "Выполнить консольную команду");

        private final String key;
        private final String description;
    }

    private long fileDirMessageId;

    //endregion

    //region AUTOWIRED and CONSTRUCTORS

    @Autowired
    private FileManager fileManager;

```

```

public TelegramBot(BotConfig botConfig) {
    this.botConfig = botConfig;
    this.botMenu = getBotMenu();
}

/**
 * Заполняет меню бота командами, считывая их из BotMenuEnum
 *
 * @return коллекцию команд
 */
private List<BotCommand> getBotMenu() {
    List<BotCommand> botMenu = new ArrayList<>();
    for (var botCommand : BotMenuEnum.values()) {
        botMenu.add(new BotCommand(botCommand.getKey(), botCommand.getDescription()));
    }
    try {
        this.execute(new SetMyCommands(botMenu, new BotCommandScopeDefault(), null));
    } catch (TelegramApiException e) {
        log.error("Не удалось добавить команды: " + e.getMessage());
    }
    return botMenu;
}

//endregion

//region OVERRIDE TelegramBots + фильтр на владельца

@Override
public String getBotUsername() {
    return this.botConfig.getBotName();
}

@Override

```

```

public String getBotToken() {
    return this.botConfig.getToken();
}

@Override
public void onUpdateReceived(Update update) {
    if (!ownerFilter(update)) return;
    if (update.hasMessage() && update.getMessage().hasText()) {           //Если прилетел текст
        handlerText(update);
    } else if (update.hasCallbackQuery()) {                               //Если прилетел
CallbackQuery
        handlerCallBackQuery(update);
        //todo можно добавить сохранение присланных файлов, но придется для каждого типа файла
(музыка, видео и т.п.) делать отдельный метод
    } else {
        log.error("Не могу распознать отправленную информацию: " + update);
    }
}

/**
 * Фильтр для идентификации владельца
 *
 * @param update объект обновления
 * @return true - обратился владелец
 */
private boolean ownerFilter(Update update) {
    boolean result;
    try {
        Long ownerId = botConfig.getOwnerId();
        result = (update.getMessage() != null && update.getMessage().getFrom().getId().equals(ownerId)) ||
                (update.getCallbackQuery() != null && update.getCallbackQuery().getFrom().getId().equals(ownerId));
    } catch (Exception e) {

```

```

        log.error("Не могу определить владельца: " + e.getMessage());
        result = false;
    }
    return result;
}

//endregion

//region Обработчики команд со слешем

/**
 * Обработчик текстовых сообщений
 *
 * @param update объект обновления
 */
private void handlerText(Update update) {
    String messageText = update.getMessage().getText();

    if (messageText.equals(BotMenuEnum.START.getKey())) {
        commandStartHandler(update);

    } else if (messageText.equals(BotMenuEnum.SHOWHOME.getKey()) || messageText.equals(ReplyKeyboardEnum.ROW1_BUTTON1.key)) {
        commandShowhomeHandler(update);

    } else if (messageText.equals(BotMenuEnum.GOTO.getKey())) {
        commandGotoHandler(update);

    } else if (messageText.equals(BotMenuEnum.SCREENSHOT.getKey())) {
        commandScreenshotHandler(update);

    } else if (messageText.equals(BotMenuEnum.MYDATA.getKey())) {
        commandMydataHandler(update);
    }
}

```

```

    } else if (messageText.equals(BotMenuEnum.DELETEDATA.getKey())) {
        commandDeletedataHandler(update);

    } else if (messageText.equals(BotMenuEnum.HELP.getKey())) {
        commandHelpHandler(update);

    } else if (messageText.equals(BotMenuEnum.SETTINGS.getKey()) || messageText.equals(ReplyKey-
boardEnum.ROW1_BUTTON2.key)) {
        commandSettingsHandler(update);

    } else if (messageText.equals(BotMenuEnum.RESET.getKey())) {
        commandResetHandler(update);

    } else if (messageText.equals(BotMenuEnum.EXIT.getKey())) {
        commandExitHandler(update);

    } else {
        //варианты команд со слешем, но с аргументами
        if (isCommand(update, OtherCommandEnum.GOTO)) {
            commandGotoArgsHandler(update);

        } else if (isCommand(update, OtherCommandEnum.SETPASS)) {
            commandSetPassHandler(update);

        } else if (isCommand(update, OtherCommandEnum.SETPMAIL)) {
            commandSetPersonalMailHandler(update);

        } else if (isCommand(update, OtherCommandEnum.SETOMAIL)) {
            commandSetOtherMailHandler(update);

        } else if (isCommand(update, OtherCommandEnum.RENAME)) {
            commandRenameHandler(update);

        } else if (isCommand(update, OtherCommandEnum.MOVE)) {

```

```

        commandMoveHandler(update);

    } else if (isCommand(update, OtherCommandEnum.COPY)) {
        commandCopyHandler(update);

    } else if (isCommand(update, OtherCommandEnum.CMD)) {
        commandCMDHandler(update);

    } else {
        commandOrElseHandler(update);
    }
}

/**
 * Проверяет, является ли сообщение командой из списка других команд
 *
 * @param update объект обновления
 * @param command команда для проверки - введенный пользователем текст
 * @return true - команда из списка других команд
 */
boolean isCommand(Update update, OtherCommandEnum command) {
    if (update.getMessage().getText().length() < command.key.length()) return false;
    return update.getMessage().getText().startsWith(command.key);
}

//endregion

//region Обработчики конкретных команд со слешем

/**
 * Обработчик команды /start
 *
 * @param update объект обновления

```



```

    */
private void commandStartHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.findUserOrRegister(update);
    sendMessageWithReplyKeyboard(chatId, report);
}

/**
 * Обработчик команды /showdir
 *
 * @param update объект обновления
 */
private void commandShowhomeHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    FileDir fileDir = fileManager.getFileDirHomeWithScan();
    sendMessageWithInlineKeyboard(
        chatId,
        "%s: %s".formatted("Текущий путь ", fileDir.getFdPath()),
        fileDir
    );
}

/**
 * Обработчик команды /goto
 *
 * @param update объект обновления
 */
private void commandGotoHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    sendMessageWithoutKeyboard(
        chatId,
        "Для того, чтоб напрямую перейти по адресу, введите команду (без кавычек и фигурных скобок) в формате '" + OtherCommandEnum.GOTO.key + " {путь для перехода}'"
    );
}

```

```

}

/**
 * Обработчик команды /goto с параметром (путём для перехода)
 *
 * @param update объект обновления
 */
private void commandGotoArgsHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String targetPath = update.getMessage().getText()
        .substring(OtherCommandEnum.GOTO.key.length()).trim();
    FileDir fileDir = fileManager.getFileDirWithScan(targetPath);
    if (fileDir == null) {
        String report = "Путь не найден: " + targetPath;
        log.error(report);
        sendMessageWithoutKeyboard(chatId, report);
        return;
    }
    sendMessageWithInlineKeyboard(
        chatId,
        "%s: %s".formatted("Текущий путь ", fileDir.getFdPath()),
        fileDir
    );
}

/**
 * Обработчик команды /setpass
 *
 * @param update объект обновления
 */
private void commandSetPassHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.updatePackPassword(
        update,

```

```

        update.getMessage().getText()
            .substring(OtherCommandEnum.SETPASS.key.length())
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Обработчик команды /setpmail
 *
 * @param update объект обновления
 */
private void commandSetPersonalMailHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.updatePersonalMail(
        update,
        update.getMessage().getText()
            .substring(OtherCommandEnum.SETPMAIL.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Обработчик команды /setomail
 *
 * @param update объект обновления
 */
private void commandSetOtherMailHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.updateOtherMail(
        update,
        update.getMessage().getText()
            .substring(OtherCommandEnum.SETOMAIL.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

```

```

}

/**
 * Обработчик команды /screenshot
 *
 * @param update объект обновления
 */
private void commandScreenshotHandler(Update update) {
    String report;
    String screenPath = fileManager.printScreen();
    if (screenPath == null) {
        report = "Не удалось создать скриншот.";
        log.error(report);
        sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
        return;
    }
    report = "Скриншот сделан. " + sendFile(update.getMessage().getChatId(), screenPath);
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
}

/**
 * Обработчик команды /mydata
 *
 * @param update объект обновления
 */
private void commandMydataHandler(Update update) {
    String report = fileManager.getUserInfo(update);
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
}

/**
 * Обработчик команды /deletedata
 *
 * @param update объект обновления

```

```

    */
private void commandDeletedataHandler(Update update) {
    String report = fileManager.deleteUser(update);
    //todo добавить полную очистку чата и оставить кнопку /start
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
}

/**
 * Обработчик команды /help
 *
 * @param update объект обновления
 */
private void commandHelpHandler(Update update) {
    //todo доделать этот метод
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), "Инструкция пока пишется.");
}

/**
 * Обработчик команды /settings
 *
 * @param update объект обновления
 */
private void commandSettingsHandler(Update update) {
    String sep = System.lineSeparator();
    String report = fileManager.getUserSettings(update) +
        "----" + sep +
        "Для изменения пароля введите команду (без кавычек и фигурных скобок)" + sep +
        "в формате: '" + OtherCommandEnum.SETPASS.key + " {новый пароль}'." + sep +
        "Например: " + OtherCommandEnum.SETPASS.key + " 1111" + sep +
        "----" + sep +
        "Для изменения личной почты (для получения на неё писем) введите команду (без кавычек"
и фигурных скобок)" + sep +
        "в формате '" + OtherCommandEnum.SETPMAIL.key + " {новый адрес личной почты}'." + sep
+

```

```

        "Например: " + OtherCommandEnum.SETPMAIL.key + " mynew@personal.mail" + sep +
        "---" + sep +
        "Для изменения другой почты (для отправки на неё писем) введите команду (без кавычек и
        фигурных скобок)" + sep +
        "в формате '" + OtherCommandEnum.SETOMAIL.key + " {новый адрес другой почты}'." + sep
+
        "Например: " + OtherCommandEnum.SETOMAIL.key + " mynew@other.mail" + sep;
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
}

/**
 * Обработчик команды /reset
 *
 * @param update объект обновления
 */
private void commandResetHandler(Update update) {
    String report = "Перезапускаю бота, подождите 5-10 секунд...";
    sendMessageWithoutKeyboard(update.getMessage().getChatId(), report);
    log.info(report);
    fileManager.botReset();
}

/**
 * Обработчик команды /exit
 *
 * @param update объект обновления
 */
private void commandExitHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = "Всё, выключаю бота на том устройстве";
    sendMessageWithoutKeyboard(chatId, report);
    log.info(report);
    System.exit(0);
}

```

```

//варианты с дробью, но с аргументами

/**
 * Обработчик команды OtherCommandEnum.RENAME
 *
 * @param update объект обновления
 */
private void commandRenameHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.fileDirRename(
        fileManager.getFileDir(),
        update.getMessage().getText()
            .substring(OtherCommandEnum.RENAME.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Обработчик команды OtherCommandEnum.MOVE
 *
 * @param update объект обновления
 */
private void commandMoveHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.fileDirMove(
        fileManager.getFileDir(),
        update.getMessage().getText()
            .substring(OtherCommandEnum.MOVE.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**

```

```

    * Обработчик команды OtherCommandEnum.COPY
    *
    * @param update объект обновления
    */
private void commandCopyHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.fileDirCopy(
        fileManager.getFileDir(),
        update.getMessage().getText()
            .substring(OtherCommandEnum.COPY.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Обработчик команды OtherCommandEnum.CMD
 *
 * @param update объект обновления
 */
private void commandCMDHandler(Update update) {
    long chatId = update.getMessage().getChatId();
    String report = fileManager.terminalExecute(
        update.getMessage().getText()
            .substring(OtherCommandEnum.CMD.key.length()).trim()
    );
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Обработчик команд не из списка
 *
 * @param update объект обновления
 */
private void commandOrElseHandler(Update update) {

```



```

        long chatId = update.getMessage().getChatId();
        sendMessageWithoutKeyboard(chatId, "Мне неизвестно что делать с командой: " +
update.getMessage().getText());
    }

//endregion

//region Обработчики Callback

/**
 * Обработчик команд Callback
 *
 * @param update объект обновления
 */
private void handlerCallBackQuery(Update update) {
    String callbackData = update.getCallbackQuery().getData();
    long messageId = update.getCallbackQuery().getMessage().getMessageId();
    long chatId = update.getCallbackQuery().getMessage().getChatId();

    // если callbackData - один из пунктов словаря актуальной папки
    if (fileManager.getCurrentPathDict().containsKey(callbackData)) {
        FileDir targetFileDir = fileManager.getCurrentPathDict().get(callbackData);

        //если нажали на Inline-кнопку МЕНЮ
        if (targetFileDir.getFdNameInline().equals(fileManager.getMENU())) {
            sendMessageWithInlineFileDirMenu(
                chatId,
                FileManager.SELECT_MENU_ITEM,
                FileManager.FileDirMenu.values()
            );
            return;
        }

        //если нажали на кнопку на файл или папку
    }
}

```

```

        targetFileDir = fileManager.getFileDirWithScan(targetFileDir.getFdPath());
        editMessageWithInlineKeyboard(
            chatId,
            messageId,
            "%s: %s".formatted("Текущий путь ", targetFileDir.getFdPath()),
            targetFileDir
        );
        return;
    }

    //если нажали на кнопку "Получить информацию"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.GET_INFO)) {
        sendMessageWithoutKeyboard(
            chatId,
            fileManager.getFileDir().toString()
        );
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    //если нажали на кнопку "Получить в Telegram"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.GET_ON_TELEGRAM)) {
        String report;
        if (fileManager.getFileDir().getFdType().equals(FileDir.FDType.FILE)) {
            report = sendFile(chatId, fileManager.getFileDir().getFdPath());
        } else {
            report = "Это папка, для отправки в Telegram, её сначала нужно упаковать в архив.";
            log.warn(report);
        }
        sendMessageWithoutKeyboard(chatId, report);
        deleteMessageWithFileDirMenu(chatId);
        return;
    }
}

```

```

//если нажали на кнопку "Получить на почту"
if (isCallbackData(callbackData, FileManager.FileDirMenu.GET_ON_EMAIL)) {
    String report = fileManager.sendEmailPersonal(
        fileManager.getFileDir(),
        update
    );
    sendMessageWithoutKeyboard(chatId, report);
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали на кнопку "Отправить на почту"
if (isCallbackData(callbackData, FileManager.FileDirMenu.SEND_TO_EMAIL)) {
    String report = fileManager.sendEmailOther(
        fileManager.getFileDir(),
        update
    );
    sendMessageWithoutKeyboard(chatId, report);
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали на кнопку "Упаковать в zip"
if (isCallbackData(callbackData, FileManager.FileDirMenu.PACK)) {
    String report = fileManager.zipFileDirWithoutPassword(
        fileManager.getFileDir()
    );
    sendMessageWithoutKeyboard(chatId, report);
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали на кнопку "Упаковать в zip с паролем"
if (isCallbackData(callbackData, FileManager.FileDirMenu.PACK_WITH_PASSWORD)) {

```

```

        String report = fileManager.zipFileDirWithPassword(
            fileManager.getFileDir(),
            update
        );
        sendMessageWithoutKeyboard(chatId, report);
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    //если нажали "Распаковать из zip"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.UNPACK)) {
        String report = fileManager.unzipFileDirWithoutPassword(
            fileManager.getFileDir()
        );
        sendMessageWithoutKeyboard(chatId, report);
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    //если нажали "Распаковать из zip с паролем"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.UNPACK_WITH_PASSWORD)) {
        String report = fileManager.unzipFileDirWithPassword(
            fileManager.getFileDir(),
            update
        );
        sendMessageWithoutKeyboard(chatId, report);
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    //если нажали на кнопку "Переименовать"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.RENAME)) {
        sendMessageWithoutKeyboard(
            chatId,

```

```

        "Для переименования текущего файла/папки введите команду (без кавычек и фигурных
скобок) в формате '" + OtherCommandEnum.RENAME.key + " {новое имя}'"
    );
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали кнопку "Переместить"
if (isCallbackData(callbackData, FileManager.FileDirMenu.MOVE)) {
    sendMessageWithoutKeyboard(
        chatId,
        "Для перемещения текущего файла/папки введите команду (без кавычек и фигурных ско-
бок) в формате '" + OtherCommandEnum.MOVE.key + " {новый путь к папке для перемещения}'"
    );
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали кнопку "Копировать"
if (isCallbackData(callbackData, FileManager.FileDirMenu.COPY)) {
    sendMessageWithoutKeyboard(
        chatId,
        "Для копирования текущего файла/папки введите команду (без кавычек и фигурных ско-
бок) в формате '" + OtherCommandEnum.COPY.key + " {новый путь к папке для копирования}'"
    );
    deleteMessageWithFileDirMenu(chatId);
    return;
}

//если нажали на кнопку "Удалить"
if (isCallbackData(callbackData, FileManager.FileDirMenu.DELETE)) {
    String report = fileManager.fileDirDelete(
        fileManager.getFileDir()
    );
}

```

```

        sendMessageWithoutKeyboard(chatId, report);
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    //если нажали на кнопку "Убрать меню"
    if (isCallbackData(callbackData, FileManager.FileDirMenu.REMOVE_MENU)) {
        deleteMessageWithFileDirMenu(chatId);
        return;
    }

    String report = "Не могу распознать нажатую кнопку " + callbackData;
    log.error(report + callbackData);
    sendMessageWithoutKeyboard(chatId, report);
}

/**
 * Метод для проверки нажатой кнопки на callbackData
 * @param callbackData нажатая кнопка
 * @param fileDirMenu меню callbackData-констант
 * @return true если нажатая кнопка - callbackData из меню
 */
private boolean isCallbackData(String callbackData, FileManager.FileDirMenu fileDirMenu) {
    boolean result = false;
    try {
        result = (!callbackData.isBlank())
            && (!fileDirMenu.getButtonCallback().isBlank())
            && callbackData.equals(fileDirMenu.getButtonCallback());
    } catch (Exception e) {
        log.error("Не удалось распознать нажатую кнопку. " + System.lineSeparator() + e.getMessage());
    }
    return result;
}

```

```

//endregion

//region ОТПРАВКА И ИЗМЕНЕНИЕ СООБЩЕНИЙ

//1. ПРОСТАЯ ОТПРАВКА СООБЩЕНИЯ

/**
 * Метод для отправки текстового сообщения без вызова клавиатур
 *
 * @param chatId      Id чата получателя
 * @param textMessage текстовое сообщение
 */
private void sendMessageWithoutKeyboard(long chatId, String textMessage) {
    try {
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textMessage);
        executeMessage(message);
    } catch (Exception e) {
        log.error("Не удалось отправить сообщение: " + e.getMessage());
    }
}

//2. ОТПРАВКА СООБЩЕНИЯ С КЛАВИАТУРОЙ

/**
 * Метод для отправки текстового сообщения с вызовом Reply-клавиатуры
 *
 * @param chatId      Id чата получателя
 * @param textMessage текстовое сообщение
 */
private void sendMessageWithReplyKeyboard(long chatId, String textMessage) {
    try {

```

```

        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textMessage);

        ReplyKeyboardMarkup keyboardMarkup = getReplyKeyboardMarkup();
        message.setReplyMarkup(keyboardMarkup);

        executeMessage(message);
    } catch (Exception e) {
        log.error("Не удалось отправить сообщение: " + e.getMessage());
    }
}

//3. ОТПРАВКА СООБЩЕНИЯ С ИНЛАЙН-КЛАВИАТУРОЙ

/**
 * Метод для отправки текстового сообщения с вызовом Inline-клавиатуры (на основе элемента файло-
вой системы)
 *
 * @param chatId      Id чата получателя
 * @param textMessage текстовое сообщение
 * @param fileDir      элемент файловой системы, содержащий информацию для создания клавиатуры
(названия кнопок, команды и т.п.)
 */
private void sendMessageWithInlineKeyboard(long chatId, String textMessage, FileDir fileDir) {
    try {
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textMessage);
        InlineKeyboardMarkup markupInLine = getInlineKeyboardMarkup(fileDir);
        message.setReplyMarkup(markupInLine);

        executeMessage(message);
    } catch (Exception e) {

```



```

        log.error("Не удалось отправить сообщение: " + e.getMessage());
    }
}

/**
 * Метод для отправки текстового сообщения с вызовом Inline-клавиатуры (на основе переданного мас-
 * сива из именованных констант)
 *
 * @param chatId          Id чата получателя
 * @param textMessage     текстовое сообщение
 * @param fileDirMenuSortedCallbacks массив из именованных констант, содержащий как Callback-ко-
 * манды, так и названия кнопок
 *
 * (предоставляет класс файлового менеджера)
 */
private void sendMessageWithInlineFileDirMenu(long chatId, String textMessage,
FileManager.FileDirMenu[] fileDirMenuSortedCallbacks) {
    try {
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(chatId));
        message.setText(textMessage);
        InlineKeyboardMarkup markupInLine = getInlineKeyboardMarkup(fileDirMenuSortedCallbacks);
        message.setReplyMarkup(markupInLine);

        this.fileDirMessageId = executeMessage(message);
    } catch (Exception e) {
        log.error("Не удалось отправить сообщение: " + e.getMessage());
    }
}

/**
 * Метод для отправки подготовленного сообщением в чат
 *
 * @param message подготовленное сообщение
 */

```

```

private Integer executeMessage(SendMessage message) {
    Integer messageId = null;
    try {
        messageId = execute(message).getMessageId();
        log.info("Сообщение с id " + messageId + " в чат " + message.getChatId() + " отправлено");
    } catch (TelegramApiException e) {
        log.error("Не удалось отправить сообщение в чат: " + e.getMessage());
    }
    return messageId;
}

//4. ОТПРАВКА ФАЙЛА

/**
 * Метод для отправки файла в чат
 *
 * @param chatId Id чата получателя
 * @param filePath путь к отправляемому файлу
 * @return отчет о выполнении
 */
private String sendFile(long chatId, String filePath) {
    String report;
    try {
        execute(new SendDocument(String.valueOf(chatId), new InputFile(new File(filePath))));
        report = "Файл " + filePath + " отправлен в чат " + chatId;
        log.info(report);
    } catch (Exception e) {
        report = "Не удалось отправить файл " + filePath + " в чат " + chatId;
        log.error(report + System.lineSeparator() + e.getMessage());
    }
    return report;
}

//5. ИЗМЕНЕНИЕ УЖЕ ОТПРАВЛЕННОГО СООБЩЕНИЯ

```

```

// 1ая перегрузка

/**
 * Метод для изменения уже отправленного сообщения без добавления клавиатур
 *
 * @param chatId      Id чата получателя
 * @param messageId   Id сообщения
 * @param textMessage текстовое сообщение для замены старого сообщения
 */
private void editMessageWithoutKeyboard(long chatId, long messageId, String textMessage) {
    try {
        EditMessageText message = new EditMessageText();
        message.setChatId(String.valueOf(chatId));
        message.setText(textMessage);
        message.setMessageId((int) messageId);

        execute(message);
        log.info("Сообщение с id " + messageId + " в чате " + chatId + " изменено");
    } catch (Exception e) {
        log.error("Не удалось изменить сообщение с id " + messageId + " в чате: " +
e.getMessage());
    }
}

// 2ая перегрузка

/**
 * Метод для изменения уже отправленного сообщения с добавлением Inline-клавиатуры
 *
 * @param chatId      Id чата получателя
 * @param messageId   Id сообщения
 * @param textMessage текстовое сообщение для замены старого сообщения
 * @param fileDir      элемент файловой системы, содержащий информацию для создания клавиатуры
(названия кнопок, команды и т.п.)

```

```

    */
    private void editMessageWithInlineKeyboard(long chatId, long messageId, String textMessage,
        FileDir fileDir) {
        try {
            EditMessageText message = new EditMessageText();
            message.setChatId(String.valueOf(chatId));
            message.setText(textMessage);
            message.setMessageId((int) messageId);

            InlineKeyboardMarkup markupInLine = getInlineKeyboardMarkup(fileDir);
            message.setReplyMarkup(markupInLine);

            execute(message);
            log.info("Сообщение с id " + messageId + " в чате " + chatId + " изменено");
        } catch (Exception e) {
            log.error("Не удалось изменить сообщение с id " + messageId + " в чате: " + e.getMes-
            sage());
        }
    }

    //6. УДАЛЕНИЕ УЖЕ ОТПРАВЛЕННОГО СООБЩЕНИЯ С ИНЛАЙН-КЛАВИАТУРОЙ

    /**
     * Метод для удаления уже отправленного сообщения с добавлением Inline-клавиатуры
     * @param chatIdLong Id чата
     */
    private void deleteMessageWithFileDirMenu(long chatIdLong) {
        Integer messageId = null;
        try {
            messageId = (int) this.fileDirMessageId;
            String chatId = String.valueOf(chatIdLong);
            DeleteMessage deleteMessage = new DeleteMessage(chatId, messageId);
            execute(deleteMessage);
            log.info("Сообщение с id " + messageId + " в чате " + chatId + " удалено");
        }
    }

```

```

        } catch (TelegramApiException e) {
            log.error("Не удалось удалить сообщение с id " + messageId + " в чате: " + e.getMessage());
        }
    }

    //endregion

    //region КЛАВИАТУРЫ

    //СОЗДАНИЕ И ВОЗВРАТ Inline-КЛАВИАТУРЫ.
    //      1ая перегрузка - для файловой системы

    /**
     * Метод создания и возврата Inline-клавиатуры на основе переданной fileDir.
     * ВАЖНО - единое правило на все команды CallbackData.
     *
     * @param fileDir элемент файловой системы, содержащий информацию для создания клавиатуры (названия кнопок, команды и т.п.)
     * @return Inline-клавиатура
     */
    private InlineKeyboardMarkup getInlineKeyboardMarkup(FileDir fileDir) {
        InlineKeyboardMarkup iKeyboard = new InlineKeyboardMarkup();
        try {
            List<List<InlineKeyboardButton>> iRows = new ArrayList<>();
            for (var inlineKeyboardIdsRow : fileDir.getFdInlineKeyboardIds()) {
                List<InlineKeyboardButton> iRow = new ArrayList<>();
                for (var inlineKeyboardId : inlineKeyboardIdsRow) {
                    InlineKeyboardButton iButton = new InlineKeyboardButton();
                    FileDir fileDirInFolder = fileManager.getCurrentPathDict().get(inlineKeyboardId);
                    iButton.setText(fileDirInFolder.getFdNameInline());
                    iButton.setCallbackData(fileDirInFolder.getFdCallbackData());
                    iRow.add(iButton);
                }
            }
        }
    }

```

```

        iRows.add(iRow);
    }
    iKeyboard.setKeyboard(iRows);
} catch (Exception e) {
    log.error("Не удалось создать Inline-клавиатуру: " + e.getMessage());
}
return iKeyboard;
}

//      2ая перегрузка - для меню элементов файловой системы

/**
 * Метод создания и возврата Inline-клавиатуры на основе переданного массива из именованных кон-
 * стант.
 *
 * @param fileDirMenuSortedCallbacks массив из именованных констант, содержащий как Callback-ко-
 * манды, так и названия кнопок
 *
 *                                     (предоставляет класс файлового менеджера)
 * @return Inline-клавиатура
 */
private InlineKeyboardMarkup getInlineKeyboardMarkup(FileManager.FileDirMenu[]
fileDirMenuSortedCallbacks) {
    InlineKeyboardMarkup iKeyboard = new InlineKeyboardMarkup();
    try {
        List<List<InlineKeyboardButton>> iRows = new ArrayList<>();

        for (FileManager.FileDirMenu fileDirMenuSortedCallback : fileDirMenuSortedCallbacks) {
            List<InlineKeyboardButton> iRow = new ArrayList<>();
            InlineKeyboardButton iButton = new InlineKeyboardButton();
            iButton.setText(fileDirMenuSortedCallback.getButtonText());
            iButton.setCallbackData(fileDirMenuSortedCallback.getButtonCallback());
            iRow.add(iButton);
            iRows.add(iRow);
        }
    }
}

```

```

        iKeyboard.setKeyboard(iRows);
    } catch (Exception e) {
        log.error("Не удалось создать Inline-клавиатуру: " + e.getMessage());
    }
    return iKeyboard;
}

/**
 * Метод создания и возврата Reply-клавиатуры по умолчанию
 *
 * @return Reply-клавиатура
 */
private ReplyKeyboardMarkup getReplyKeyboardMarkup() {
    return getReplyKeyboardMarkup(this.replyKeyboardNames);
}

/**
 * Метод создания и возврата Reply-клавиатуры
 *
 * @param replyKeyboardNames названия кнопок
 * @return Reply-клавиатура
 */
private ReplyKeyboardMarkup getReplyKeyboardMarkup(List<List<String>> replyKeyboardNames) {
    ReplyKeyboardMarkup rKeyboard = new ReplyKeyboardMarkup();
    try {
        List<KeyboardRow> rRows = new ArrayList<>();
        for (var replyKeyboardNamesRow : replyKeyboardNames) {
            KeyboardRow rRow = new KeyboardRow();
            for (var rName : replyKeyboardNamesRow) {
                rRow.add(rName);
            }
            rRows.add(rRow);
        }
        rKeyboard.setKeyboard(rRows);
    }
}

```

```
        //rKeyboard.setOneTimeKeyboard(true);
        rKeyboard.setResizeKeyboard(true);
    } catch (Exception e) {
        log.error("Не удалось создать Reply-клавиатуру: " + e.getMessage());
    }
    return rKeyboard;
}

//endregion
}
```


Приложение 20. Файл pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in-
stance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>ru.dogobot</groupId>
    <artifactId>Dogobot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Dogobot</name>
    <description>Virtual friend assistant Dogobot for command execution </description>
    <properties>
        <java.version>17</java.version>
        <telegram.version>6.8.0</telegram.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
```

```

        <groupId>org.telegram</groupId>
        <artifactId>telegrambots</artifactId>
        <version>${telegram.version}</version>
    </dependency>
    <dependency>
        <groupId>com.vdurmont</groupId>
        <artifactId>emoji-java</artifactId>
        <version>5.1.1</version>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- API для автоматизирования сопоставления между XML-документами и объектами Java.
        Без этого не работало -->
    <dependency>
        <groupId>javax.xml.bind</groupId>
        <artifactId>jaxb-api</artifactId>
        <version>2.3.1</version>
    </dependency>

    <!--         <dependency>-->
    <!--         <groupId>org.springframework.boot</groupId>-->
    <!--         <artifactId>spring-boot-starter-mail</artifactId>-->
    <!--         </dependency>-->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-email</artifactId>

```

```

        <version>1.5</version>
    </dependency>
    <dependency>
        <groupId>net.lingala.zip4j</groupId>
        <artifactId>zip4j</artifactId>
        <version>2.11.5</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
                <executable>true</executable> <!-- Чтоб jar в target стал исполняемым -->
            </configuration>
        </plugin>
    </plugins>
</build>

</project>

```


Приложение 21. Папка resources, файл logback.xml

```
<configuration>

  <property name="HOME_LOG" value="../logs/dogo.log"/>
<!--      <property name="HOME_LOG" value="..\logs\dogo.log"/>-->

  <appender name="FILE-ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${HOME_LOG}</file>

    <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
      <fileNamePattern>../logs/dogo.%d{yyyy-MM-dd}.%i.log.gz</fileNamePattern>
      <!-- each archived file, size max 10MB -->
      <maxFileSize>10MB</maxFileSize>
      <!-- total size of all archive files, if total size > 20GB, it will delete old archived
file -->
      <totalSizeCap>1GB</totalSizeCap>
      <!-- 60 days to keep -->
      <maxHistory>60</maxHistory>
    </rollingPolicy>

    <encoder>
      <pattern>%d %p %c{1} [%t] %m%n</pattern>
    </encoder>
  </appender>

  <logger name="ru.dogobot.Dogobot" level="debug" additivity="false">
    <appender-ref ref="FILE-ROLLING"/>
  </logger>

  <root level="error">
    <appender-ref ref="FILE-ROLLING"/>
  </root>
```

```
<root level="info">  
    <appender-ref ref="FILE-ROLLING"/>  
</root>  
  
</configuration>
```

Приложение 22. Папка resources, файл application.properties

Собирается из 5 файлов.

Файл **application.properties**:

```
spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden1.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden1.properties
```

Файл **.hidden1.properties**:

```
spring.application.name=Dogobot
spring.main.headless=false

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden2.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden2.properties
```

Файл **.hidden2.properties**:

```
telegrambot.name=botdogobot
telegrambot.token=[токен, полученный для бота]
telegrambot.authorId=[id автора программы]
telegrambot.ownerId=[id владельца программы]

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden3.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden3.properties
```

Файл **.hidden3.properties**:

```
spring.datasource.url=[jdbc:postgresql-путь к базе данных]
spring.datasource.username=[username для входа в базу данных]
spring.datasource.password=[пароль для входа в базу данных]
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.config.import=/home/username/archive/myproject/Dogobot/.hidden/.hidden4.properties
#spring.config.import=D:\\archive\\myproject\\Dogobot\\.hidden\\.hidden4.properties
```

Файл **.hidden4.properties**:

```
email.smtp.host=[сервер исходящей почты]
email.smtp.port=[порт исходящей почты]
email.imap.host=[сервер входящей почты]
email.imap.port=[порт входящей почты]
email.from=[почтовый адрес с которого будут отправляться письма]
email.password=[пароль от почтового адреса]
```

Файл **.hidden5.json**:

```
{
  "PACK_PASSWORD_KEY": "пароль для упаковки и распаковки файлов и папок",
  "PERSONAL_EMAIL_KEY": "адрес персональной почты",
  "OTHER_EMAIL_KEY": "дополнительный адрес другой почты"
}
```