СОДЕРЖАНИЕ

ВВЕДЕНИЕ 2

1 ОБЗОР ЛИТЕРАТУРЫ 4

1.1 Взаимодействие c Telegram 4

1.2 Взаимодействие с Team City 6

1.3 Обзор аналогов 6

1.4 Выбор используемых технологий 6

1.5 Выводы 6

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 6

2.1 Структура программного средства 7

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 7

3.1 Описание работы фреймворка 7

3.3 Описание структуры и взаимодействия между классами 7

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 8

4.1 Алгоритм 8

4.2 Алгоритм 8

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ 9

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 11

6.1 Установка системы на удаленном сервере 11

6.2 Описание основных функций программного средства 11

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА 12

7.1 Характеристика программного продукта 12

7.2 Расчет сметы затрат, цены и прибыли 12

7.3 Расчет экономического эффекта у пользователя 12

ЗАКЛЮЧЕНИЕ 14

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 15

ПРИЛОЖЕНИЕ А 16

ПРИЛОЖЕНИЕ Б 17

ПРИЛОЖЕНИЕ В 18

ПРИЛОЖЕНИЕ Г 19

# ВВЕДЕНИЕ

С постоянным развитием информационных технологий, увеличиваются требования к производимому продукту и его сложность. С увеличением сложности продукта, растет и ответственность за организацию работы. Крупный проект требует вовлечение большего количества людей, тем самым затрудняя взаимодействие. По этим причинам создаются множества программных средств, нацеленных на решение некоторых из вышеописанных проблем. Но желание распрощаться со всеми проблемами сразу не всегда приводит к эффективным решениям. Множество требующихся программ увеличивает время подготовки рабочего места, а также время на обучение использованию каждой из них.

Данный дипломный проект представляет собой программное средство, основанное на мэссенджере Telegram, которое облегчает использование системы непрерывной интеграции TeamCity и системы отслеживания задач Atlassian JIRA с помощью непосредственного взаимодействия с ними через чат, а также неявного их взаимодействия между собой.

Перед каждой ИТ-компанией, в ходе её функционирования, становится вопрос распределения задач между работниками, а также отслеживание выполнения этих самых задач, потраченного на задачу времени, отслеживание ошибок и статуса их исправления.

Управление задачами – это процесс постановки задач исполнителям, промежуточный и итоговый контроль их выполнения. Каждая задача – это шаг для достижения целей компании. Поэтому важно, чтобы все задачи выполнялись качественно и в срок.

Многое усложняет работу руководителя. Могут изменяться сроки, приоритеты и важность задач, а также сами задачи. Эта часть работы руководителя, которая отнимает много времени и сил. Если отсутствует система управления задачами, работа многократно усложняется. Однако и при использовании такой системы существует необходимость следить за тем, чтобы работники не забывали взаимодействовать с системой: отмечать завершенность задач, указывать затраченное время, давать задачам статус и другое.

Система управления задачами – это набор инструментов для постановки задач и контроля над их выполнением. В данном дипломном проекте будет использоваться в качестве такой системы Atlassian JIRA Software. JIRA Software — это мощная система для отслеживания задач, обладающая невероятно широкими возможностями персонализации. Каждая команда разработчиков по-своему подходит к организации рабочего процесса, поэтому в JIRA Software есть множество готовых шаблонов. Система масштабируема и подходит как для организаций с небольшим количеством сотрудников (менее 10 человек), так и для очень крупных предприятий (до 10000 человек).

Еще одним немаловажным инструментом команды разработчиков является система непрерывной интеграции. Непрерывная интеграция — это практика разработки программного обеспечения, которая заключается в слиянии рабочих копий в общую основную ветвь разработки несколько раз в день и выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем. В обычном проекте, где над разными частями системы разработчики трудятся независимо, стадия интеграции является заключительной. Она может непредсказуемо задержать окончание работ. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий.

Система отслеживания задач и система непрерывной интеграции являются важными инструментами команды разработчиков. Но каждая система требует отдельного взаимодействия, что значительно сказывается на временных затратах. Сборка крупного проекта на сервере непрерывной интеграции также часто является длительным процессом. Чтобы объединить взаимодействие с данными системами, а также реализовать уведомления разработчиков об окончании сборки, было решено разработать программное средство на основе средства мгновенного обмена сообщениями.

Средство мгновенного обмена сообщениями, мессенджер — программное средство и протокол передачи данных, позволяющие интернет-пользователям общаться в реальном времени. Обычно мессенджеры включают в себя программы-клиенты, устанавливаемые на локальных компьютерах. Популярными мессенджерами являются: ICQ (работает по протоколу OSCAR), Skype (собственный закрытый проприетарный протокол), Telegram (основан на открытом программном обеспечении). Telegram является одним из самых безопасных и популярных сервисов мгновенного обмена сообщениями. Сквозное шифрование трафика здесь является опциональным и включается только если пользователь сам этого пожелает, выбрав функцию "Создать секретный чат" (что, вполне оправдано, учитывая, что порядка 90% трафика не содержит конфиденциальной информации, и шифровать её по умолчанию смысла нет). Кроме того, в секретных чатах блокируется снятие скриншотов с экрана, а на сообщения можно установить таймер самоуничтожения. Также программа имеет открытые исходники, что позволяет убедиться в отсутствии у нее каких-либо недокументированных функций.

В ходе дипломного проектирования планируется разработать программное средство для удаленного управления сборкой приложений и взаимодействия с системой отслеживания задач со следующими функциями:

- возможность запустить сборку проекта из чата;

- отправка уведомлений о результатах сборки проекта в групповой чат (например, успешно или неуспешно и список изменений, кто менял);

- закрытие задач в JIRA, которые указаны в сообщениях коммитов;

- подсвечивание ссылками номеров задач JIRA в текстах новостей о билдах;

- уведомления об изменения статуса задач в чате;

- назначение и изменение параметров задачи в чате (отработанное время, статус).

# **1** ОБЗОР ЛИТЕРАТУРЫ

## **1.1** Взаимодействие c Telegram

Telegram — бесплатный [кроссплатформенный](https://ru.wikipedia.org/wiki/%25D0%259A%25D1%2580%25D0%25BE%25D1%2581%25D1%2581%25D0%25BF%25D0%25BB%25D0%25B0%25D1%2582%25D1%2584%25D0%25BE%25D1%2580%25D0%25BC%25D0%25B5%25D0%25BD%25D0%25BD%25D0%25BE%25D0%25B5_%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25BD%25D0%25BE%25D0%25B5_%25D0%25BE%25D0%25B1%25D0%25B5%25D1%2581%25D0%25BF%25D0%25B5%25D1%2587%25D0%25B5%25D0%25BD%25D0%25B8%25D0%25B5) [мессенджер](https://ru.wikipedia.org/wiki/%25D0%25A1%25D0%25B8%25D1%2581%25D1%2582%25D0%25B5%25D0%25BC%25D0%25B0_%25D0%25BC%25D0%25B3%25D0%25BD%25D0%25BE%25D0%25B2%25D0%25B5%25D0%25BD%25D0%25BD%25D0%25BE%25D0%25B3%25D0%25BE_%25D0%25BE%25D0%25B1%25D0%25BC%25D0%25B5%25D0%25BD%25D0%25B0_%25D1%2581%25D0%25BE%25D0%25BE%25D0%25B1%25D1%2589%25D0%25B5%25D0%25BD%25D0%25B8%25D1%258F%25D0%25BC%25D0%25B8) для смартфонов и других устройств, позволяющий обмениваться текстовыми сообщениями и медиафайлами различных форматов. Количество активных пользователей сервиса на февраль 2016 года составляло более 100 млн человек, а количество ежедневно пересылаемых сообщений достигло 10 миллиардов на август 2015 [1].

При помощи специального API сторонние разработчики могут создавать «ботов», специальные аккаунты, управляемые программами, созданные для того, чтобы автоматически обрабатывать и отправлять сообщения. По сути, эти аккаунты играют роль интерфейса к сервису, который работает на удалённом сервере. 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.

Для создания бота следует написать пользователю @BotFather. BotFather — один бот, чтобы править всеми. При помощи него меняются настройки у существующих ботов и создаются новые. Чтобы создать нового робота следует написать команду **/**newbot. Далее BotFather спросит имя (name) нового бота и username. Имя (name) будет отображаться в контактах и чатах. Username — короткое имя на латинице, которое используется для упоминаний бота. Username должен состоять из букв латинского алфавита, подчёркиваний и цифр и быть длиной от  5 до 32 символов. Также имя пользователя обязательно должно заканчиваться на «bot», например: «tetris\_bot» или «TetrisBot». После этого BotFather предоставит ключ авторизации (токен). Ключ авторизации это набор символов вида 110201543:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw, который нужен, чтобы получать и отправлять сообщения с помощью Bot API [2].

Bot API представляет из себя HTTP-интерфейс для работы с ботами в Telegram. Все запросы к Telegram Bot API должны осуществляться через HTTPS в следующем виде:

https://api.telegram.org/bot<token>/НАЗВАНИЕ\_МЕТОДА.

Допускаются GET и POST запросы. Для передачи параметров в Bot API доступны 4 способа:

* Запрос в URL
* application/x-www-form-urlencoded
* application/json (не подходит для загрузки файлов)
* multipart/form-data (для загрузки файлов)

Ответ придёт в виде JSON-объекта, в котором всегда будет булево поле ok и опциональное строковое поле description, содержащее человекочитаемое описание результата. Если поле ok истинно, то запрос прошёл успешно и результат его выполнения можно увидеть в поле result. В случае ошибки поле ok будет равно false, а причины ошибки будут описаны в поле description. Кроме того, в ответе будет целочисленное поле error\_code, но коды ошибок будут изменены в будущем.

Существует два диаметрально противоположных по логике способа получать обновления от вашего бота: длинные опросы (long polling) и вебхуки (webhooks). Независимо от способа получения обновлений, в ответ вы получите объект Update, сериализованный в JSON. Этот объект представляет из себя входящее обновление. Под обновлением подразумевается действие, совершённое с ботом — например, получение сообщения от пользователя. Принципиальное отличие: при long polling приложению самому нужно запрашивать обновления у API, а используя вебхуки — сервера Telegram будут отправлять на сервер каждое обновление с помощью HTTPS POST-запроса [3].

Длинные опросы (long poll). Другое название способа — "очередь ожидающих запросов». Способ заключается в периодическом опросе серверов Telegram на предмет наличия новой информации. Схема работы такова (рисунок 1) [4]:

1) Клиент отсылает запрос на сервер.

2) Сервер, вместо того, чтобы быстро обработать этот запрос и отправить ответ клиенту, запускает цикл, в каждой итерации которого следит за возникновением событий (другой клиент добавил запись или удалил).

3) При возникновении события сервер генерирует ответ и отсылает его клиенту, таким образом завершая запрос.

4) Клиент, получив ответ от сервера, запускает обработчик события и параллельно отправляет очередной «длинный» запрос серверу.

Рисунок 1.

У данного способа есть недостаток: если на сервере одновременно запущено несколько ботов, вероятность столкнуться с ошибками возрастает.

Webhook — механизм оповещения пользователей системы о событиях. Используя вебхуки — сервера Telegram будут отправлять на сервер каждое обновление с помощью HTTPS POST-запроса. Отпадает необходимость периодически опрашивать серверы. Однако появляется необходимость установки полноценного веб-сервера на ту машину, на которой планируется запускать ботов. Также необходимо иметь собственный SSL-сертификат, т.к. вебхуки в Telegram работают только по HTTPS.

Исходя из всего вышеперечисленного, было принято решение использовать способ длинных опросов (long polling).

Для взаимодействия с Telegram будет использоваться библиотека с открытым исходным кодом, реализованная на языке программирования Java TelegramBots [5], основанная на Telegram Bot API. Для использования библиотеки следует с помощью фреймворка Apache Maven добавить следующую зависимость:

<dependency>

<groupId>org.telegram</groupId>

<artifactId>telegrambots</artifactId>

<version>2.4.4</version>

</dependency>

Для создания бота, основанного на длинных опросах, следует наследовать собственный класс от класса TelegramLongPollingBot. В свою очередь, для создания бота, использующего вебхуки, следует наследовать от TelegramWebhookBot. Класс, наследующий TelegramLongPollingBot, должен реализовывать 3 метода:

* public void onUpdateReceived(Update update). Этот метод будет вызываться всякий раз при получении обновление Update (например, в случае получения сообщения, адресованного боту);
* public String getBotUsername(). Этот метод должен всегда возвращать username бота;
* public String getBotToken(). Этот метод должен всегда возвращать токен бота, полученный у BotFather.

После этого остается проинициализировать контекст API при помощи ApiContextInitializer.init(), создать TelegramBotsApi и зарегистрировать своего бота с помощью метода registerBot() объекта TelegramBotsApi.

## **1.2** Взаимодействие с Team City

## **1.3** Обзор аналогов

## **1.4** Выбор используемых технологий

## **1.5** Выводы

# **2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Разрабатываемый проект необходимо разбить на отдельные логически взаимосвязанные модули, что является необходимым условием для обеспечения гибкости структуры программного модуля. При данном подходе становится возможной выборочная модернизация отдельных частей программного кода, с минимальным влиянием на остальные части проекта, либо, в идеальном случае, вовсе без их изменения.

Для определения логических модулей необходимо в первую очередь определить основные возможности программного модуля:

* возможность авторизации в социальных сетях, определенных ранее;
* возможность сбора данных из социальных сетей;
* анализ собранных данных различными способами;
* дополнение собранных данных о местоположении пользователя географическими координатами;
* возможность импорта дополнительных данных для облегчения задачи анализа;
* экспорт полученных результатов для дальнейшего использования;
* визуализация данных путем генерации таблиц и графиков;
* формирование отчетов по результатам анализа данных;

## **2.1** Структура программного средства

# **3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Рассмотрим подробным образом функционирование разрабатываемого программного модуля.

## **3.1** Описание работы фреймворка

**3.2** Описание модели данных

**3.2.1** Таблица

**3.2.2** Таблица

**3.2.3** Таблица

## **3.3** Описание структуры и взаимодействия между классами

Изучение модели данных, используемых в программном продукте, является одним из основных этапов понимания процесса обработки программой данных что, как известно, является основной конечной целью любой разрабатываемой программной системы. Не менее значимым этапом с этой точки зрения является описание структуры классов и взаимодействия между ними. Для описания структуры классов перечислим основные классы программного модуля с указанием способов их взаимодействия между собой, методов и полей, используя диаграмму классов (см. чертеж ГУИР.400201.003 РР.1) в соответствии с модулями, выделенными на этапе системного проектирования (см. чертеж ГУИР.400201.003 С1).

**3.3.1** Класс

Методы класса:

**3.3.2** Класс

Поля класса:

Методы класса:

**3.3.3** Класс

# **4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программного модуля анализа данных профессиональных социальных сетей использовалось значительное количество сторонних библиотек, что объясняется необходимостью выполнения определенного перечня стандартных задач по работе с API сторонних сервисов, анализу данных и генерации отчетов. Следует, однако, заметить, что социальные сети и иные сервисы схожего назначения в открытый доступ посредством API выносят лишь небольшую часть данных, доступных при помощи веб-интерфейса. Данные особенности работы с данными оформлены в виде специальных соглашений об использовании данных и недвусмысленно упоминают о запрете на автоматизацию обращений к веб-версии сайта с целью получения недостающей части данных. Следует заметить, что данные, доступные публично через удобные для разработчика API, ожидаемо являются в некоторой степени полезными для задач анализа, однако значительно более интересные свойства и характеристики аккаунтов закрыты для использования. При создании настоящего программного модуля ставилась задача получения наиболее интересных видов данных о пользователях сервисов любыми средствами, в том числе и нарушающими пользовательские соглашения и соглашения об использовании данных. Рассмотрим некоторые алгоритмы, позволяющие организовать получение данных, их дальнейший анализ и отображение. Полный листинг наиболее специфических участков программного кода приводится в приложении к настоящей пояснительной записке (см. приложение А).

## **4.1** Алгоритм

## **4.2** Алгоритм

# **5** ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Одной из наиболее значимых задач в ходе жизненного цикла разработки ПО является тестирование. Тестирование программного обеспечения — это процесс исследования ПО с целью выявления ошибок и определения соответствия между реальным и ожидаемым поведением ПО, осуществляемый на основе набора тестов, выбранных определённым образом. В более широком смысле, тестирование ПО — это техника контроля качества программного продукта, включающая в себя проектирование тестов, выполнение тестирования и анализ полученных результатов.

Каждый этап разработки системы сопровождался написанием большого количества разнообразных тестов. Каждая новая функциональность программы тщательно проверялась различными методами. В ходе разработки были применены такие виды тестирования, как:

* функциональное тестирование;
* тестирование производительности;
* тестирование стабильности;
* тестирование совместимости;
* тестирование интерфейса пользователя.

На сегодняшний день промышленные способы проверки качества программного обеспечения используют различные автоматизированные системы тестирования. В Ruby on Rails такой системой по умолчанию является TestUnit. Данная автоматизированный модуль позволяет проводить тестирование моделей, контроллеров и представлений. В среду Rails встроена на удивление надежная и готовая к использованию система тестирования. Затрачивая минимум усилий, можно задавать воспроизводимые настройки баз данных, отправлять Web-приложениям тестовые HTTP-сообщения и выполнять три вида тестирования: модульное, функциональное и комплексное.

При написании тестов для различных уровней приложения значительно снижаются риски нарушения работоспособности приложения при внесении различных изменений в существующий функционал. Полное покрытие тестами занимает порядка 38% от общего времени работы программиста над проектом, но временные затраты полностью окупаются эффектом надежности и стабильности программного обеспечения. По мере продвижения проекта стоимость устранения дефектов ПО может экспоненциально возрастать. Инструменты статического и динамического анализа помогают предотвратить эти затраты благодаря обнаружению программных ошибок на ранних этапах жизненного цикла ПО.

При написании данной дипломной работы разработка велась по принципу TDD (Test Driven Development), реализующему процесс написания приложения через тестирование. Основополагающий принцип TDD – это написание тестов до написания кода приложения.

Разработка через [тестирование](https://ru.wikipedia.org/wiki/%252525D0%252525A2%252525D0%252525B5%252525D1%25252581%252525D1%25252582%252525D0%252525B8%252525D1%25252580%252525D0%252525BE%252525D0%252525B2%252525D0%252525B0%252525D0%252525BD%252525D0%252525B8%252525D0%252525B5_%252525D0%252525BF%252525D1%25252580%252525D0%252525BE%252525D0%252525B3%252525D1%25252580%252525D0%252525B0%252525D0%252525BC%252525D0%252525BC%252525D0%252525BD%252525D0%252525BE%252525D0%252525B3%252525D0%252525BE_%252525D0%252525BE%252525D0%252525B1%252525D0%252525B5%252525D1%25252581%252525D0%252525BF%252525D0%252525B5%252525D1%25252587%252525D0%252525B5%252525D0%252525BD%252525D0%252525B8%252525D1%2525258F)  — техника [разработки программного обеспечения](https://ru.wikipedia.org/wiki/%252525D0%252525A0%252525D0%252525B0%252525D0%252525B7%252525D1%25252580%252525D0%252525B0%252525D0%252525B1%252525D0%252525BE%252525D1%25252582%252525D0%252525BA%252525D0%252525B0_%252525D0%252525BF%252525D1%25252580%252525D0%252525BE%252525D0%252525B3%252525D1%25252580%252525D0%252525B0%252525D0%252525BC%252525D0%252525BC%252525D0%252525BD%252525D0%252525BE%252525D0%252525B3%252525D0%252525BE_%252525D0%252525BE%252525D0%252525B1%252525D0%252525B5%252525D1%25252581%252525D0%252525BF%252525D0%252525B5%252525D1%25252587%252525D0%252525B5%252525D0%252525BD%252525D0%252525B8%252525D1%2525258F), которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам.  Разработка через тестирование требует от разработчика создания автоматизированных [модульных тестов](https://ru.wikipedia.org/wiki/%252525D0%2525259C%252525D0%252525BE%252525D0%252525B4%252525D1%25252583%252525D0%252525BB%252525D1%2525258C%252525D0%252525BD%252525D0%252525BE%252525D0%252525B5_%252525D1%25252582%252525D0%252525B5%252525D1%25252581%252525D1%25252582%252525D0%252525B8%252525D1%25252580%252525D0%252525BE%252525D0%252525B2%252525D0%252525B0%252525D0%252525BD%252525D0%252525B8%252525D0%252525B5), определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, либо нет. Когда они выполняются, говорят, что тест пройден.

Тестирование в рамках TDD позволяет получить простой способ оценки полноты интерфейсов: необходимым и достаточным считается такой интерфейс, который позволяет выполнить все написанные тесты. Все, что находится за этими рамками, считается ненужным. Наконец, использование тестов в качестве инструмента дизайна заставляет программиста в первую очередь концентрироваться на интерфейсе, а уже во вторую - на имплементации, что также положительно влияет на результат.

# **6** РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

## **6.1** Установка системы на удаленном сервере

## **6.2** Описание основных функций программного средства

# **7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

## **7.1** Характеристика программного продукта

## **7.2** Расчет сметы затрат, цены и прибыли

## **7.3** Расчет экономического эффекта у пользователя

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Telegram (мессенджер) [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Telegram\_(мессенджер)](https://ru.wikipedia.org/wiki/Telegram_(%25D0%25BC%25D0%25B5%25D1%2581%25D1%2581%25D0%25B5%25D0%25BD%25D0%25B4%25D0%25B6%25D0%25B5%25D1%2580)).
2. Роботы [Электронный ресурс]. – Режим доступа: <https://tlgrm.ru/docs/bots>
3. Документация Telegram API [Электронный ресурс]. – Режим доступа: <https://tlgrm.ru/docs/bots/api>
4. Длинные опросы (long polling) [Электронный ресурс]. – Режим доступа: <http://javascript.ru/ajax/comet/long-poll>
5. TelegramBots: Java library to create bots using Telegram Bot API [Электронный ресурс]. – Режим доступа: <https://github.com/rubenlagus/TelegramBots>

# ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код некоторых алгоритмов программного модуля

# ПРИЛОЖЕНИЕ Б

(обязательное)

# ПРИЛОЖЕНИЕ В

(обязательное)

Спецификация программного дипломного проекта

# ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов