Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Курсовой проект по дисциплине:

«Программирование»

Пояснительная записка к курсовой работе

Тема работы:

«Бот-секретарь для лабораторных работ»

Исполнитель

студент гр. 653502 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шилов М.Г.

(подпись дата)

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Козуб В.Н.

(подпись дата)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка)

Минск

2017 год

**Содержание**

[Введение 3](#_Toc480311999)

[1. Telegram 4](#_Toc480312000)

[1.1 Описание мессенджера 4](#_Toc480312001)

[1.2 Telegram боты 4](#_Toc480312002)

[1.3 Создание бота 5](#_Toc480312003)

[2. Ruby 6](#_Toc480312004)

[2.1 Описание языка 6](#_Toc480312005)

[2.2 Взаимодействие с Bot API 7](#_Toc480312007)

[3. Логика бота 9](#_Toc480312008)

[3.1 Команды 9](#_Toc480312009)

[3.2.1 ООП 10](#_Toc480312014)

[3.2.2 База данных 11](#_Toc480312016)

[3.2.3 Структура программы 11](#_Toc480312019)

[3.3 Описание работы бота 13](#_Toc480312020)

[4. Heroku 15](#_Toc480312021)

[4.1 Описание работы системы 15](#_Toc480312022)

[4.2 Создание приложения на Heroku 16](#_Toc480312023)

[4.3 Развёртывание бота 16](#_Toc480312024)

[Выводы 18](#_Toc480312025)

[Список использованных источников 19](#_Toc480312026)

## Введение

Толчком к написанию данной курсовой работы послужило увеличившееся количество предметов с лабораторными работами в университете. В связи с чем стало сложнее планировать своё свободное время и распределять его для выполнения лабораторных работ по каждому из предметов.

Например, в этом семестре у меня есть три предмета с лабораторными работами: «Основы алгоритмизации и программирования» (7 лабораторных работ), «Программирование» (7 лабораторных работ) и «Инструменты и средства программирования» (8 лабораторных работ). Возникает вопрос: в каком порядке и количестве необходимо делать лабораторные работы?

Целью данной курсовой работы является разработка бота, который будет сообщать о количестве лабораторных работ по каждому предмету, которые должны быть сданы к текущему моменту для успешного окончания семестра.

Я решил использовать Telegram в качестве платформы для своего бота по нескольким причинам: я регулярно пользуюсь им в качестве своего основного мессенджера, а также Telegram предоставляет API для ботов, что упрощает разработку.

В качестве языка программирования используется Ruby.

Для того, чтобы бот работал круглосуточно, используется платформа Heroku.

## 1. Telegram

## 1.1 Описание мессенджера

**Telegram** — бесплатный кроссплатформенный мессенджер для смартфонов и других устройств, позволяющий обмениваться текстовыми сообщениями и медиафайлами различных форматов. Используются проприетарная серверная часть c закрытым кодом, работающая на мощностях нескольких компаний США и Германии, и несколько клиентов с открытым исходным кодом, в том числе под GNU GPL.

Учётные записи пользователей привязываются к телефонным номерам, что является одним из самых существенных аргументов критиков Telegram, поскольку это не обеспечивает полной анонимности при общении. При регистрации в сервисе и последующих авторизациях новых устройств, производится проверка телефонного номера через отправку SMS-сообщения с кодом (на некоторых операционных системах — перехватывается приложением) или телефонный вызов.

Количество активных пользователей сервиса на февраль 2016 года составляло более 100 млн человек, а количество ежедневно пересылаемых сообщений достигло 10 миллиардов на август 2015.

Для мессенджера был создан протокол MTProto, предполагающий использование нескольких протоколов шифрования. При авторизации и аутентификации используются алгоритмы RSA-2048, DH-2048 для шифрования, при передаче сообщений протокола в сеть они шифруются AES с ключом, известным клиенту и серверу. Также применяются криптографические хеш-алгоритмы SHA-1 и MD5.

Безопасность от перехвата пересылаемых сообщений со стороны сервера Telegram обеспечивается лишь в режиме «секретных» чатов (Secret Chats), доступном с 8 октября 2013 года. Этот режим реализует шифрование, при котором лишь отправитель и получатель обладают общим ключом (end-to-end шифрование), с применением алгоритма AES-256 в режиме IGE (англ. Infinite Garble Extension) для пересылаемых сообщений. В отличие от обычного режима, сообщения в секретных чатах не расшифровываются сервером, история переписки сохраняется лишь на тех двух устройствах, на которых был создан чат.

При обмене файлами можно как отправить файлы с устройства, так и искать медиаконтент в интернете, в том случае, если используется мобильная версия для iOS или Android. Размер передаваемых файлов ограничен 1,5 Гб. Программа использует систему докачки файлов после обрыва связи.

Имеется возможность организовывать мультичаты до 200 участников, начиная с ноября 2015 года, супергруппы до 1000 участников, с 14 марта 2016 — супергруппы до 5000 участников.

## 1.2 Telegram боты

При помощи специального API (Bot 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.

Боты — особые аккаунты, которые не требуют номера телефона при создании. По сути, эти аккаунты играют роль интерфейса к сервису, который работает на удалённом сервере.

Отличия ботов от обычных аккаунтов:

* У ботов нет статусов «онлайн» и «был в сети», вместо этого отображается надпись «bot».
* Для ботов выделено ограниченное место на серверах — все сообщения будут удалены по прошествии определённого срока после обработки.
* Боты не могут сами начать общение с пользователем. Пользователь должен либо добавить бота в группу, либо первым начать с ним диалог. Для этого можно использовать ссылки вида telegram.me/<bot\_username> или поиск по имени пользователя.
* Имя пользователя у робота должно заканчиваться на «bot» (например, [@TriviaBot](https://telegram.me/triviabot), [@gadalka\_bot](https://telegram.me/gadalka_bot)).

## 1.3 Создание бота

Для того, чтобы создать бота в Telegram, необходимо написать боту [@BotFather](https://telegram.me/botfather) и следовать его инструкциям.

Отправляем ему команду «/newbot», которая создаёт нового бота. В ответ на эту команду [@BotFather](https://telegram.me/botfather) просит нас ввести имя бота, которое будет служить его адресом в Telegram (telegram.me/<имя\_бота>). Данное имя должно быть уникальным и оканчиваться на «bot». Создадим бота с именем EasyLabBot – данное имя свободно. [@BotFather](https://telegram.me/botfather) сообщает нам, что бот успешно создан и доступен по ссылке telegram.me/EasyLabBot. Также он отправляет нам HTTP API token, который необходим для доступа к нашему боту.

Однако, если отправить сообщение нашему боту, то никакого ответа не последует, так как мы не прописали в него никакой логики. Логику бота будем писать на Ruby

## 2. Ruby

## 2.1 Описание языка

## Ruby — динамический, рефлективный, интерпретируемый высокоуровневый язык программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу. Кроссплатформенная реализация интерпретатора языка является полностью свободной

Целью разработки Ruby было создание «настоящего объектно-ориентированного», лёгкого в разработке, интерпретируемого языка программирования.

Создатель Ruby — Юкихиро Мацумото, фанат объектно-ориентированного программирования, мечтал о языке, более мощном, чем Perl, и более объектно-ориентированном, чем Python. Основное назначение Ruby — создание простых и в то же время понятных программ, где важна не скорость работы программы, а малое время разработки, понятность и простота синтаксиса.

Язык следует принципу «наименьшей неожиданности»: программа должна вести себя так, как ожидает программист. Однако в контексте Ruby это означает наименьшее удивление не при знакомстве с языком, а при его основательном изучении.

Сам Мацумото утверждает, что целью разработки была минимизация неожиданностей при программировании для него, но после распространения языка он с удивлением узнал, что мышление программистов похоже, и для многих из них принцип «наименьшей неожиданности» совпал с его принципом.

Ruby также унаследовал идеологию языка программирования Perl в части предоставления программисту возможностей достижения одного и того же результата несколькими различными способами. Люди различны, и им для свободы необходима возможность выбирать. «Я предпочитаю обеспечить много путей, если это возможно, но поощрять или вести пользователей, чтобы выбрать лучший путь, если это возможно».

Одной из основных целей разработки было освобождение программистов от рутинной работы, которую вычислитель может выполнять быстрее и качественнее. Особое внимание, в частности, уделялось будничным рутинным занятиям (обработка текстов, администрирование), и для них язык настроен особенно хорошо.

В противовес машинно-ориентированным языкам, работающим быстрее, целью этой разработки был язык, наиболее близкий к человеку. Любая работа с компьютером выполняется людьми и для людей, и необходимо заботиться в первую очередь о затрачиваемых усилиях людей. Язык позволяет максимально быстро и просто для человека выполнить задачу, хотя, возможно, это и потребует дополнительного времени работы компьютера.

Ruby — полностью объектно-ориентированный язык. В нём все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы. Каждая функция — метод.

Ruby не поддерживает множественное наследование, но вместо него есть мощный механизм примесей. Все классы (напрямую или через другие классы) выведены из класса Object, следовательно, любой объект может использовать определённые в нём методы (например, class, to\_s, nil?). Процедурный стиль также поддерживается, но все глобальные процедуры неявно являются закрытыми методами класса Object.

Ruby является мультипарадигменным языком: он поддерживает процедурный стиль (определение функций и переменных вне классов), объектно-ориентированный (всё — объект), функциональный (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат функцией последнего вычисленного значения). Он поддерживает рефлексию, метапрограммирование, информацию о типах переменных на стадии выполнения.

Работа с массивами — одна из сильных сторон Ruby. Они автоматически изменяют размер, могут содержать любые элементы, а язык предоставляет мощные средства для их обработки.

Как и в большинстве языков программирования, в Ruby можно использовать широкий набор сторонних библиотек. Большая часть из них реализована в форме гема. RubyGems – менеджер пакетов Ruby, созданный для упрощения процесса создания, распространения и установки библиотек. Основное место, где хранятся библиотеки - RubyGems.org, предоставляет библиотеки Ruby в виде гемов. Можно найти необходимые гемы прямо на сайте или воспользоваться командой gem.

## 2.2 Взаимодействие с Bot API

Telegram предоставляет 2 вида API: Bot API (для написания ботов) и Telegram API (для создания собственных модифицированных клиентов для Telegram). Для внедрения логики нашего бота нам необходимо использовать Bot API. Bot API представляет из себя HTTP-интерфейс для работы с ботами. Однако вместо того, чтобы вручную прописывать все HTTP-запросы рациональнее будет использовать gem «[telegram-bot-ruby](https://github.com/atipugin/telegram-bot-ruby)**» − обёртка для Bot API,** является промежуточным слоем между прикладной программой и API.

Пример использования:

require 'telegram/bot'

token = 'YOUR\_TELEGRAM\_BOT\_API\_TOKEN'

Telegram::Bot::Client.run(token) do |bot|

bot.listen do |message|

case message.text

when '/start'

bot.api.send\_message(chat\_id: message.chat.id, text: "Привет, #{message.from.first\_name}")

when '/stop'

bot.api.send\_message(chat\_id: message.chat.id, text: "Пока, #{message.from.first\_name}")

end

end

end

Вместо YOUR\_TELEGRAM\_BOT\_API\_TOKEN необходимо подставить token, который прислал @BotFather при создании бота. Сохраним данный текст в файл с названием «bot.rb» и запустим командой «ruby bot.rb».

Теперь отправим нашему боту команду «/start». Если всё было сделано правильно, он ответит «Привет, ВАШЕ\_ИМЯ». Если мы отправим команду «/stop», то он ответит «Пока, ВАШЕ\_ИМЯ».

Итак, мы выяснили, как заставить бота отвечать на команды. Теперь нужно адаптировать это под нашу задачу.



## 3. Логика бота

## 3.1 Команды

* + 1. Основная задача бота напоминать студентам, какие предметы и сколько лабораторных работ должно быть сдано к сегодня, чтобы не давать им расслабляться. Для этого предстоит научить его следующим командам:
* /start **−** выводит приветствие и описание всех доступных команд
* /semester **−** запоминает даты начала и конца семестра
* /subject **−** добавляет предмет и количество лабораторных работ по нему
* /status **−** выводит список лабораторных работ студента, которые ему предстоит сдать
* /reset **−** сбрасывает для пользователя все данные.

Бот на основании введенных пользователем данных о начале и конце семестра, сегодняшней дате и количеству лабораторных работ должен предполагать, сколько лабораторных работ студенту надо было бы уже сдать.

**Примеры работы команд:**

> /start

< Привет. Я тебе смогу помочь сдать все лабораторные работы, чтобы родители не ругались. Смотри список того, что я умею:

\* /semester - запоминает даты начала и конца семестра

\* /subject - добавляет предмет и количество лабораторных работ по нему

\* .....

> /semester

< Когда начинаем учиться?

> 09.02.2017

< Когда надо сдать все лабораторные работы?

> 07.06.2017

> Понял, на все про все у нас 3 месяца и 29 дней.

> /subject

< Какой предмет учим?

> ОАиП

> Сколько лабораторных работ надо сдать?

< 7

> ОК

> /subject

< Какой предмет учим?

> Программирование

> Сколько лабораторных работ надо сдать?

< 7

> ОК

> /subject

< Какой предмет учим?

> ИСП

> Сколько лабораторных работ надо сдать?

< 8

> ОК

> /status

< К этому времени у тебя должно было быть сдано:

> ОАиП - 3 из 7

> Программирование - 3 из 7

> ИСП - 4 из 8

>

> Давай поднажмем!

Чтобы сделать бота более удобным в использовании, необходимо, чтобы эти команды автодополнялись при вводе:

## Снимок%20экрана%202017-04-17%20в%2017.24.44.png

## Рисунок 1. Автодополнение команды.

## Реализовать это можно, отправив @BotFather команду «/mybots», выбрав нужного бота. Далее нажать на кнопки «Edit Bot», «Edit Commands» и отправить команды в виде:

Команда1 - описание

Команда2 – Другое описание

## 3.2 Код программы

## 3.2.1 ООП

## При создании данного бота мы будем придерживаться ООП. Объектно-ориентированное программирование — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Каждый класс будет храниться в отдельном файле, что позволит легко удалять старые и добавлять новые функции в нашу программу.

## 3.2.2 База данных

## Бот должен будет запоминать предметы и их количество у каждого пользователя. Следовательно, необходима база данных для их хранения. В качестве базы данных будем использовать Redis. Redis (англ. remote dictionary server) — сетевое журналируемое хранилище данных типа «ключ — значение» с открытым исходным кодом. Нереляционная высокопроизводительная СУБД.

## Очевидными преимуществами являются простота подключения и скорость работы.

**Подключение**

Для начала необходимо запустить сервер Redis. Делается это командой «redis-server». Для использования Redis в нашем боте необходимо установить gem «redis». Сервер Redis по умолчанию запускается со стандартной конфигурацией на localhost и «прослушивает» порт 6379.

require "redis"

@redis = Redis.new(:host => "127.0.0.1", :port => 6379)

@redis.set("mykey", "hello world")

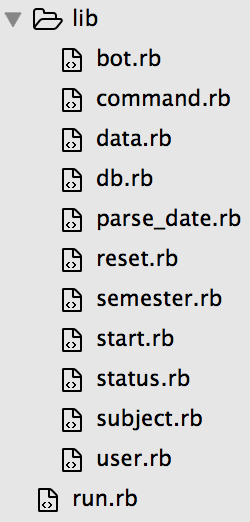
# => "OK"

@redis.get("mykey")

# => "hello world"

Создаётся объект базы данных @redis, далее по ключу «mykey» устанавливается значение «hello world».

## 3.2.3 Структура программы

1. Как уже было сказано ранее, весь код будет разбит на классы. В каждом файле будет один класс. Структура:
2. 
3. Рисунок 2. Структура программы
4. *command.rb* — файл, в котором описан класс Command. Позволяет боту отправлять сообщения. От класса Command будут наследоваться все остальные команды бота (каждая команда находится в своём классе).
5. *db.rb* — файл, в котором описан класс Database. Подключает бота к базе данных Redis. В нём определены методы для взаимодействия с базой данных.
6. *data.rb* — файл, в котором хранятся приватные данные. Например, token бота.
7. *start.rb* — файл, в котором описан класс Start. Наследуется от класса Command. Отправляет приветственное сообщение пользователю.
8. *semester.rb* — файл, в котором описан класс Semester. Наследуется от класса Command. Хранит в себе начало и конец семестра.
9. *subject.rb* — файл, в котором описан класс Subject. Наследуется от класса Command. Хранит в себе информацию о предмете и количеству лабораторных работ по нему.
10. *status.rb* — файл, в котором описан класс Status. Наследуется от класса Command. Выводит пользователю информацию о количестве лабораторных работ, которые необходимо сдать.
11. *reset.rb* — файл, в котором описан класс Reset. Наследуется от класса Command. Удаляет все данные пользователя.
12. *parse\_date.rb* — файл, в котором описан класс DateParser. В нём происходят проверки корректного ввода даты, а также вычисления количества месяцев и дней, которое осталось для сдачи всех лабораторных работ.
13. *user.rb* — файл, в котором описан класс User. Определяет поля и методы, используемые для работы с пользователем.
14. *bot.rb* — файл, подключающий все файлы команд бота. В нём описан класс TelegramBot. В данном файле содержится вся основная логика работы бота.
15. *run.rb* — файл, запускающий бота. В нём создаётся объект класса TelegramBot.

## 3.3 Описание работы бота

При запуске бот соединяется с базой данных. После этого он начинает получать абсолютно все сообщения от всех пользователей.

Чтобы различать пользователей он использует их ID в Telegram. Для того, чтобы хранить данные каждого пользователя и никак их не перепутать, они хранятся в Redis в виде хэша (словаря) — «ключ – значение». Ключом выступает ID пользователя, а значением — хэш с данными пользователя.

Когда пользователь отправляет сообщение, бот определяет ID пользователя и по этому ID получает его данные.

Далее бот определяет статус пользователя. Например, если пользователь ранее ввёл команду «/semester», то он хочет указать начало семестра. Следовательно, бот будет принимать начало семестра. Если пользователь введёт некорректные данные, бот отправит ему сообщение об ошибке. Если статус пользователя не удовлетворяет ни одному из прописанных в программе («хочу начало семестра», «хочу конец семестра», «хочу ввести предмет», «хочу ввести количество лабораторных работ»), то бот будет принимать только команды, которые были заранее определены («/start», «/semester», «/subject», «/status», «/reset»).

Таким образом, бот будет работать для каждого пользователя, с которым он заговорит.

Однако бот активен до тех пор, пока работает программа на компьютере. После того, как программа завершается, бот перестаёт обрабатывать сообщения пользователей.

Следовательно, необходимо сделать так, чтобы программа работала круглосуточно. Для этого его необходимо запустить на VPS.

**VPS** (англ. Virtual Private Server) или VDS (англ. Virtual Dedicated Server) — услуга, в рамках которой пользователю предоставляется так называемый Виртуальный выделенный сервер. В плане управления операционной системой по большей части она соответствует физическому выделенному серверу. В частности: root-доступ, собственные IP-адреса, порты, правила фильтрования и таблицы маршрутизации.

Одной из особенностей VPS является возможность установки и запуска собственных программ, что нам и нужно для запуска бота. Однако большим недостатком является высокая стоимость VPS.

К счастью, для наших целей существуют бесплатные аналоги. Например, Heroku.

## 4. Heroku

## 4.1 Описание работы системы

**Heroku** — облачная PaaS-платформа, поддерживающая ряд языков программирования. Heroku, одна из первых облачных платформ, появилась в июне 2007 года и изначально поддерживала только язык программирования Ruby, но на данный момент список поддерживаемых языков также включает в себя Java, Node.js, Scala, Clojure, Python, Go и PHP. На серверах Heroku используются операционные системы Debian или Ubuntu.

Приложения, работающие на Heroku, используют также DNS-сервер Heroku (обычно приложения имеют доменное имя вида «имя\_приложения.herokuapp.com»). Для каждого приложения выделяется несколько независимых виртуальных процессов, которые называются «dynos». Они распределены по специальной виртуальной сетке («dynos grid»), которая состоит из нескольких серверов. Heroku также имеет систему контроля версий Git.

Heroku предоставляет несколько тарифных планов:

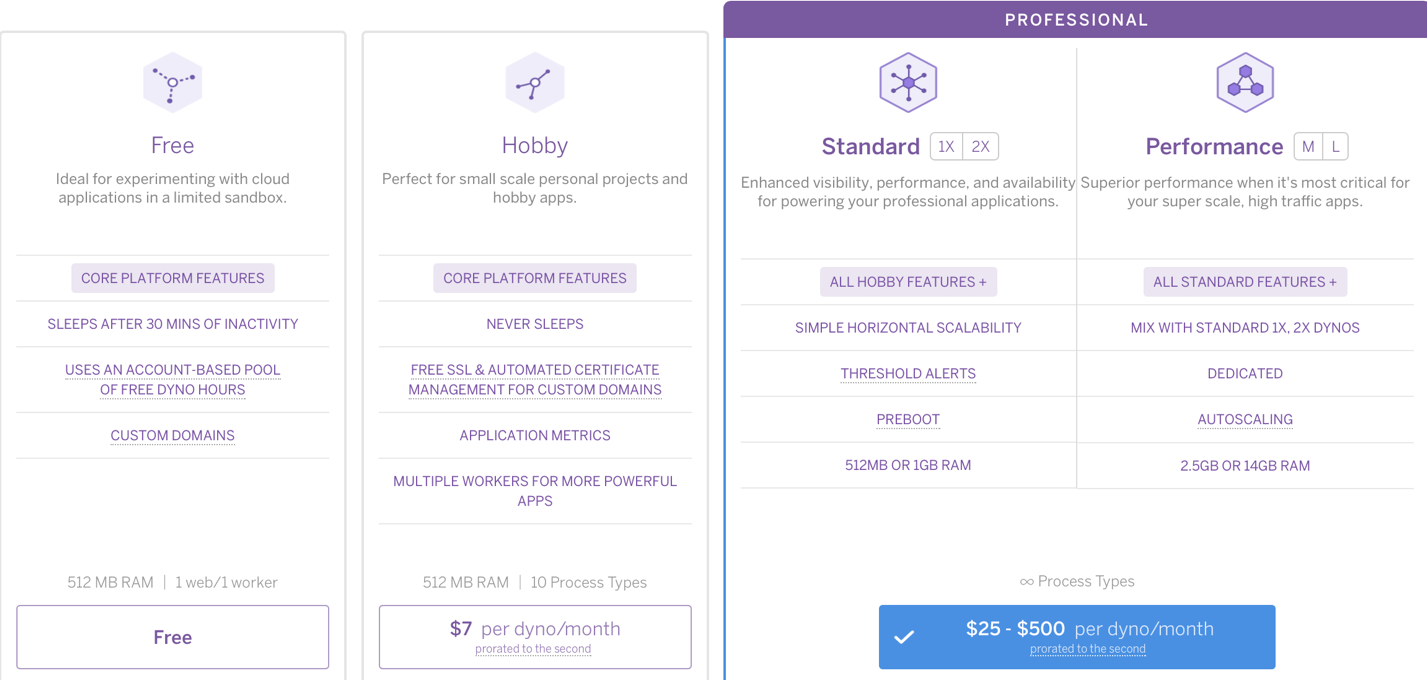


Рисунок 3. Тарифные планы Heroku

Для наших целей подойдёт тарифный план «Free», который предоставляется бесплатно.

Приложение запускается при помощи «worker». При этом расходуются «dyno hours». После регистрации в Heroku, согласно тарифному плану «Free», бесплатно представляется 600 «dyno hours» каждый месяц, то есть 600 часов работы приложения. После ввода данных банковской карты это число увеличивается до 1000. Также появляется возможность установки различных расширений (add-ons). При этом денежные средства с банковской карточки не списываются.

В Heroku, в отличие от обычного VPS хостинга предоставляется не прямой доступ к виртуальной машине с операционной системой, а только средства для публикации приложения в преднастроенной среде.

## 4.2 Создание приложения на Heroku

После регистрации и авторизации в аккаунте на Heroku, необходимо нажать на кнопку «New», затем нажать на кнопку «Create new app». Далее необходимо указать имя приложения (имя должно быть уникальным) и регион, на серверах которого будет запускаться наше приложение.

Затем необходимо установить расширение (add-on) [Heroku Redis](https://addons-sso.heroku.com/apps/f51c0a2f-c522-437f-b94a-a986bead82db/addons/9b3a24b5-1708-451e-9532-8353e1d45c38), которое необходимо для работы Redis на Heroku. После установки расширения в «Config vars» (глобальные переменные приложения Heroku) добавится переменная REDIS\_URL**.**

## 4.3 Развёртывание бота

Приложение развёртывается при помощи Heroku Git. Сперва необходимо установить Heroku CLI — Heroku Command Line Interface — средство для управления приложением с командной строки.

Далее необходимо авторизоваться при помощи команды «heroku login». Необходимо ввести логин и пароль от аккаунта Heroku.

После этого нужно скачать репозиторий существующего приложения на компьютер и исходным кодом бота. Затем добавляем в появившуюся папку «easylabbot» файлы нашего бота.

Однако прежде чем загружать файлы бота на Heroku, необходимо внести некоторые изменения.

Во-первых, необходимо создать Gemfile. Gemfile — файл, в котором прописаны все gem-зависимости нашей программы.

Содержимое Gemfile:

source "https://rubygems.org"

ruby "2.3.1"

gem "telegram-bot-ruby"

gem "redis"

gem "json"

После этого мы можем запустить команду «bundle install» и Bundler установит все gem-зависимости для нашей программы. Также он создаст файл Gemfile.lock

Далее необходимо создать Procfile. Procfile — механизм для объявления того, какие команды выполняются при помощи «dynos» на Heroku. Его можно использовать для объявления различных типов процессов (web или worker).

Содержимое Procfile:

worker: bundle exec ruby run.rb

Также необходимо в файле db.rb заменить Redis.new(host: "127.0.0.1", port: 6379) на Redis.new(url: ENV["REDIS\_URL"]), так как сервер Redis запускается уже не на localhost, а на каком-то url на Heroku, который выдаётся при установке расширения Heroku Redis и хранится в глобальной переменной REDIS\_URL.

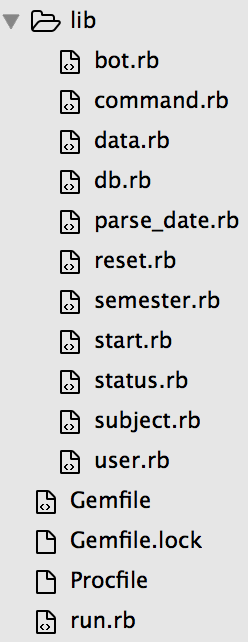


Рисунок 4. Окончательная структура программы

После этого нужно добавить все файлы нашего проекта в систему контроля версий git с помощью следующих команд:

git add .

git commit –m 'First commit'

Далее загружаем все файлы на Heroku с помощью команды git push heroku master

После того, как Heroku сообщит, что приложение было запущено можно написать нашему боту в Telegram. Если он ответит на команды, следовательно, всё работает. Если же нет, значит возникла какая-то ошибка. Найти её можно с помощью команды heroku logs, которая выведет информацию об ошибках приложения.

Итак, мы добились того, чтобы бот работал круглосуточно.

## Выводы

В данной курсовой работе был разработан бот для Telegram, ведущий учёт лабораторных работ, которые необходимо сдать к сегодняшнему дню, чтобы успешно закончить семестр.

В ходе курсового проектирования были изучены Telegram Bot API и платформа Heroku. Рассмотрены основные характеристики используемого языка программирования Ruby, а также было использовано хранилище логов Redis в качестве базы данных.

Бот был протестирован многими пользователями с использованием различных входных данных. Ошибок при этом выявлено не было.

В перспективе планируется увеличить количество пользователей данного бота. Также возможно появление новых функций, что будет несложно реализовать, так как было использовано ООП.

Также следует отметить, что в сентябре 2015 года Павел Дуров заявил о скором появлении возможностей монетизации и размещения рекламы в ботах, что делает их создание ещё более перспективным.

## Список использованных источников

1. https://ru.wikipedia.org/wiki/Telegram\_(мессенджер)
2. <https://tlgrm.ru/docs/bots>
3. <https://ru.wikipedia.org/wiki/Ruby>
4. <https://www.ruby-lang.org/ru/libraries/>
5. <https://core.telegram.org/bots/api>
6. https://ru.wikipedia.org/wiki/Обёртка\_библиотеки
7. <https://github.com/atipugin/telegram-bot-ruby>
8. <https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование>
9. <https://ru.wikipedia.org/wiki/Redis>
10. <https://github.com/redis/redis-rb>
11. <https://ru.wikipedia.org/wiki/VPS>
12. https://ru.wikipedia.org/wiki/Heroku
13. https://www.heroku.com/