МИНИСТЕРСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ»

ФАКУЛЬТЕТ ЭЛЕКТРОСВЯЗИ

КАФЕДРА ???

Разработка Telegram Bot «Курс валют Национального Банка Республики Беларусь»

Пояснительная записка

к курсовой работе

по дисциплине

«Основы разработки и сопровождения Web - приложений»

Выполнил студент гр. СО841 Гусев Н.Н.

Руководитель ???

Минск 2020

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc55515343)

[ВВЕДЕНИЕ 3](#_Toc55515344)

[1. Технологии чат ботов 4](#_Toc55515345)

[2. Проектирование и разработка Telegram Bot 6](#_Toc55515346)

[2.1 Этапы проектирования и разработки 6](#_Toc55515347)

[2.2 Обоснование выбора стека технологий 7](#_Toc55515348)

[3. Описание разработанного telegram бота 10](#_Toc55515349)

[3.1 Описание разработанного Telegram бота 10](#_Toc55515350)

[3.2 Обзор Web-приложения и его компонентов 10](#_Toc55515351)

[3.2.1 Telegram Bot 10](#_Toc55515352)

[3.2.2 Telegram Bot Ruby 11](#_Toc55515353)

[3.2.3 API Национального Банка Республики Беларусь 13](#_Toc55515354)

[3.2.4 Описание основных функций разработанного Web-приложения 15](#_Toc55515355)

[ЗАКЛЮЧЕНИЕ 19](#_Toc55515356)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc55515357)

[Приложение А – Листинг программного кода 21](#_Toc55515358)

# ВВЕДЕНИЕ

На сегодняшний день WEB – разработка занимает приоритетное направление, технологии которой способствующие покрыть задачи многих направлений. Технологии, применяемые при написании WEB – приложений и сервисов можно применить в мобильной разработке, разработке контроллеров, серверов и ботов для Telegram, WhatsApp, Вконтанке и др. социальных сетей.

Целью курсового проекта является практическое применение знаний, связанный с разработкой Telegram Bot, приобретённых в процессе обучения, а также приобретение новых умений в сфере WEB - разработки.

К основным задачам курсового проекта относится:

1. Выбор языка программирования, на котором будет написано приложение;

2. Проектирование и разработка Telegram Bot, обоснование выбора технологий;

3. Описание работы и функциональных возможностей разработанного Telegram Bot.

Объектом курсового проекта является Telegram Bot «Курс валют Национального Банка Республики Беларусь», которое в процессе реализации проекта необходимо было разработать. В наше время данное приложение является весьма актуальным, учитывая динамику курса валют на фоне мировых событий. Среди плюсов данного Telegram Bot можно отметить:

1. Актуальность и «свежесть» курса на момент запроса;

2. Конвертация любых сумм;

3. Доступность из любой точки планеты с минимальными потребностями в интернет – соединение на стороне пользователя.

# Технологии чат ботов

Современный Интернет — это универсальная среда для общения, развлечений и обучения. Общение через Сеть стало частью жизни многих людей. В настоящее время в мире существует довольно много средств, форм и способов общения, и немалая часть из них так или иначе связана с современными техническими возможностями, которые, в частности, представлены использованием глобальной компьютерной сети. Интернет, кроме полноценного источника разнообразной полезной для всех информации, также является основной формой виртуального общения. Связь с родными и близкими людьми, контакты с рабочими партнерами, новые знакомства — все это является важным компонентом повседневной жизни современного человека, причем выбор наиболее удобных способов онлайн-общения у современного пользователя достаточно большой.

Боты — это сторонние приложения, которые работают внутри Telegram. Пользователи могут взаимодействовать с ботами, отправляя им сообщения, команды и встроенные запросы. Вы управляете своими ботами с помощью HTTPS-запросов к нашему Bot API. Боты Telegram являются чат – ботами.

Чат -бот — это отдельный чат в мессенджере, в котором вы общаетесь не с человеком, а с программой. Он может присылать сообщения в виде текста, изображений, кнопок и многих других UI элементов и реагировать на сообщения от пользователей. Современные чат-боты — это полноценные UI приложения внутри мессенджеров.

При оценке ботов используется две основные группы факторов:

* Юзабилити (удобство использования);
* Создание ценности для пользователя.

Ботов можно использовать для:

* Получайте индивидуальные уведомления и новости. Бот может действовать как умная газета, рассылая вам релевантный контент, как только он будет опубликован.
* Интегрируйте с другими сервисами. Бот может дополнять чаты Telegram контентом из внешних сервисов. Gmail Bot, GIF бот , IMDB бот , Вики боты , Музыка боты , Youtube бот , GitHub бот.
* Принимайте платежи от пользователей Telegram. Бот может предлагать платные услуги или работать как виртуальная витрина.
* **Создавайте собственные инструменты. Бот** может предоставлять вам оповещения, прогнозы погоды, переводы, форматирование или другие услуги.
* Создавайте одиночные и многопользовательские игры. Бот может предложить богатые возможности HTML5, от простых аркад и головоломок до 3D-шутеров и стратегических игр в реальном времени.
* Создавайте социальные службы . Бот может связывать людей, которые ищут партнеров для разговора на основе общих интересов или близости.
* Практически все остальное. Кроме посуды - боты плохо моют посуду.

И это лишь малая часть возможностей ботов. Единственное ограничение – фантазия заказчика.

По сути, боты — это особые учетные записи, для которых не требуется дополнительный номер телефона. Пользователи могут взаимодействовать с ботами двумя способами:

* Отправляйте сообщения и команды ботам, открывая с ними чат или добавляя их в группы.
* Отправляйте запросы прямо из поля ввода, вводя @username бота и запрос. Это позволяет отправлять контент от встроенных ботов прямо в любой чат, группу или канал.

Сообщения, команды и запросы, отправленные пользователями, передаются в программное обеспечение, работающее на ваших серверах. Наш промежуточный сервер обрабатывает все шифрование и связь с Telegram API за вас. Вы общаетесь с этим сервером через простой HTTPS-интерфейс, который предлагает упрощенную версию Telegram API. Мы называем этот интерфейс нашим Bot API.

Для создания бота в Telegram вам понадобится… бот. Для этого необходимо просто поговорить с BotFather и выполните несколько простых шагов.

Реализацию логики работы бота можно реализовать на любом серверном языке программирования.

Таким образом можно сделать вывод, что в существующем мире любой пользователь может реализовать бота на свой вкус и цвет, обладая минимальными навыками программирования и алгоритмизации. Боты Telegram вывели индустрию на совершенно новый уровень, т.к. работа с ними и настройка не только просты, но и увлекательны!

# Проектирование и разработка Telegram Bot

* 1. **Этапы проектирования и разработки**

Проектирование корпоративного веб приложения, как и любого другого приложения, стоит начать с определения первоначальный целей и области решаемых задач. Создать реестр заинтересованных лиц.

На следующем этапе необходимо собрать требования к приложению, которое необходимо разработать. Уточнить цели и область решаемых задач и построить иерархическую структуру работ.

каждое web-приложение отправляет http запросы на web-сервер для получения полезных данных. Программа под управлением web-сервера использует ту или иную модель для хранения данных. В современном мире чаще всего используются базы данных, SQL или NoSQL.  
  
    Формально каждое web-приложение можно разбить на 3 взаимно независимые части.

1. Модуль, который исполняется WEB-браузером. Это приложение может быть написано на любом языке, который поддерживает браузер. Чаще всего используется язык JavaScript, как наиболее поддерживаемый и имеющий большую библиотечную поддержку. Это очень важно, так как позволяет существенно экономить бюджеты проектов.
2. Модуль, исполняемый на серверной стороне под управлением web-сервера. Это приложение может быть написано на любом языке, интерпретацию которого поддерживает выбранный Вами web-сервер. Последнее время, часто, в качестве языка программирования выбирается язык Java. Этот язык также имеет серьезную библиотечную поддержку.
3. База данных. В этой области так же существует достаточно широкий выбор. Есть промышленные базы данных, такие как Oracle, DB2, PostgreSQL. Есть легкие базы данных, такие как MySQL. База данных выбирается, основываясь на целях и области решаемых задач.

При построении архитектуры web — приложения необходимо максимально уменьшить зависимость между структурными единицами. В общем случае приложение состоит из трех структурных единиц:

1. Модуль, который работает под управлением браузера.
2. Модуль, который работает под управлением web -сервера.
3. База данных.

Для достижения цели максимальной независимости между структурными единицами, необходимо чтобы каждая структурная единица оперировала только необходимым ей набором данных. Рассмотрим более подробно.

Браузер — это прикладное программное обеспечение для просмотра web страниц.

HTML – это стандартный язык разметки документов. Большинство современных web-браузеров способны интерпретировать язык HTML.

Web сервер — это программное обеспечение, которое способно принимать HTTP запросы от клиентов, обрабатывать их и отправлять ответ в соответствии со стандартом протокола.

База данных — это представленная в объективной форме совокупность самостоятельных материалов, систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью ЭВМ.

Для минимизации зависимостей между «Браузером» и Web-сервером необходимо, чтобы язык разметки HTML был задействован только в браузере, а Web-сервер предоставлял интерфейс для получения необходимых данных для страницы.

Для решения этой задачи необходимо:

Определить цели и область решаемых задач, которые будут решаться в рамках создаваемого интерфейса.

1. Определить API серверной части.
2. Выбрать протокол взаимодействия между серверной и клиентской частью. Создание протокола удобнее всего выбрать на базе XML, так как большинство современных браузеров имеют встроенную поддержку этого языка.
3. Написать документ, в котором будет изложен протокол.
   1. **Обоснование выбора стека технологий**

Для реализации Back – End части курсового задания был выбран динамический, рефлективный, интерпретируемый высокоуровневый язык программирования Ruby.

Фактически, Front – end части, в приложении, как таковой нет. Это обусловлено использованием продуманной системы ботов Telegram.

Ruby является мощным инструментом для разработки. К его плюсам можно отнести:

1. Продуманный и очень лаконичный синтаксис, на котором код выглядит максимально читаемым. Можно писать действительно понятно и лаконично.
2. Огромное количество реализованных библиотек, подключение которых не вызывает проблем. Самый главный плюс ruby — это rails, лучший фреймворк. Четко реализованные паттерны, не то, что на node.js, крути как хочешь. Само написание кода напоминает английскую речь.

Можно сказать, что не все герои носят плащи. Ruby, изучив его плюсы, тому примеру. Для полной оценки языка программирования стоит разобрать так же и его минусы, которых, увы, не мало:

1. Требователен к ресурсам, всю историю разработки языка сопровождают серьезные оптимизации по потреблению памяти. В эталонной реализации интерпретатора (MRI) отсутствует реальная многопоточность с использованием нескольких ядер процессора (GIL).
2. Динамическая типизация. Но это сложно отнести к минусам, ведь она позволяет ускорить разработку, хоть и с потерей качества.
3. Язык практически не развивается на фоне его ближайших конкурентов (JavaScript и Python). Взрывной рост интереса к Ruby основывался в первую очередь на Ruby on Rails. Но сейчас, когда Rails-like фреймворки есть в каждом популярном ЯП, Ruby мало что может предложить.

Ruby был выбран на основе комьюнити, огромного количества качественно написанных туториалов и best practices; Rails фреймворка, делающего разработку максимально понятной от самого старта проекта, до деплоя; богатства подходов и реализаций для написания тестов: RSpec, MiniTest и так далее.

Для получения текущего курса валют был использован API Национального Банка Республики Беларусь. API реализует получения официального курса белорусского рубля по отношению к иностранным валютам, устанавливаемого Национальным банком Республики Беларусь. С сентября 2020 года Национальным Банком Республики Беларусь был осуществлен переход с HTTP на HTTPS. С момента перехода все сервисы интернет-сайта (новости в формате RSS, API, получение данных, используя XML, и др.) стали также доступны только по протоколу HTTPS.

Для связки Back – end части и Telegram было использовано BOT API Telegram и библиотека для Ruby – Telegram Bot API и HTTParty. Все запросы Telegram Bot API осуществляются через HTTPS.

Так же был предусмотрен прокси, в случае «технических проблем» у операторов и провайдеров связи и интернет, реализованный на основе библиотеки Telegram Bot API Proxy.

В качестве среды разработки использовался редактор кода Visual Studio Code с установленными плагинами.

# Описание разработанного telegram бота

## 3.1 Описание разработанного Telegram бота

Основной задачей данного Telegram бота является упрощение получения курса валют и конвертация на основе белорусского рубля.

При создании данного электронного telegram бота были проанализированы различные способы реализации задания. Как результат, путём использования данного Web-приложения несколькими лицами, был выявлен наиболее комфортный для большинства пользователей интерфейс.

## 3.2 Обзор Web-приложения и его компонентов

## 3.2.1 Telegram Bot

Telegram Bot «Курс валют Национального Банка Республики Беларусь» представлен на рисунке 3.1.

Рисунок 3.1 – Старнциа бота в Telegram

Запуск бота с использованием команды /start представлен на рисунке 3.2. По выполнению команды /start Telegram Bot «Курс валют Национального Банка Республики Беларусь» начианет работу с текущем пользвоателем. Пользователь получает сообщение приветсвия и начальные кнопки для более удобного начала работы с ним.

Рисунок 3.2 – Запуск бота в Telegram

Получение списка валют и их названий происходит при использование команды /list или при нажатие на кнопку «Достпуные курсы валют». Пример работы приведен на рисунке 3.3 текущего раздела.

Рисунок 3.3 – Доступные коды валют

Для ознакомленяи с возможностями Telegram Bot «Курс валют Национального Банка Республики Беларусь» предусмотрена команда /help. Выполнение данной команды выводит список достпунх команд внутри бота, предоставленный на рисунке 3.4.

Рисунок 3.4 – Пользовательская помощь

Получение курса валют просиходит посредтвом ввода команды /rage. Дананя команда имеет две вариации выполнения:

* /rage КОД\_ВАЛЮТЫ – курс относительно банковской единицы запрашиваемой валюты.
* /rage КОД\_ВАЛЮТЫ СУММА – курс относительно указанной суммы запрашиваемой валюты.

где КОД\_ВАЛЮТ – буквенный код валюты (USD, EUR, RUB и т.д.), СУММА – количество средств для конвертации.

Команда /rage производит конвертацию к Белорусскому рублю. Пример выполнения команды предоставлен на рисунке 3.5.



Рисунок 3.5 – Выполнение конвертации

Бот поддерживает следующие команды для взаимодейсвтия с ним:

* /start – Запуск бота. Приветствие пользователя.
* /list – поддерживаемые курсы валют (Символьные коды валюты, обслуживающиеся Национальным Баноком РБ)
* /rage КОД\_ВАЛЮТЫ СУММА – конвертация указанной валюты в белорусккий рубль по курсу Национального Банка Республики Беларусь.
* /rage КОД\_ВАЛЮТЫ – курс валюты оносительно указанного кода.
* /stop – завершенеи работы бота на текущем Telegram – аккаунте

## 3.2.2 Telegram Bot Ruby

Боты — это сторонние приложения, которые запускаются внутри Telegram. Пользователи могут взаимодействовать с ботами, отправляя им сообщения, команды и встроенные запросы. Управление ботами происходит с помощью HTTPS-запросов к Telegram Bot API.

Библиотека Telegram Bot Ruby является оболочкой Telegram Bot API для реализации [getUpdates](https://core.telegram.org/bots/api#getupdates) на стороне Back – end.

Каждый бот получает уникальный маркер проверки подлинности, когда он будет создан, называемый токеном.

Все запросы к Telegram Bot API должны быть поданы через HTTPS и должны быть представлены в следующем виде: [https://api.telegram.org/bot<token>/METHOD\_NAME](https://api.telegram.org/bot%3ctoken%3e/METHOD_NAME).

Telegram Bot API поддерживает методы GET и POST HTTP. Telegram Bot API поддерживает четыре способа передачи параметров в запросах Bot API:

* Строка запроса URL;
* application / x-www-form-urlencoded;
* application / json (кроме загрузки файлов);
* multipart / form-data (используется для загрузки файлов).

Ответ содержит объект JSON, который всегда имеет логическое поле «ok» и может иметь дополнительное поле String «description» с понятным для человека описанием результата. Если «ok» равно true, запрос был успешным, и его результат можно найти в поле «результат». В случае неудачного запроса «ok» равно false, а ошибка объясняется в «описании». Также возвращается целочисленное поле error\_code, но его содержимое может измениться в будущем. Некоторые ошибки также могут иметь необязательное поле «параметры» типа ResponseParameters, которое может помочь автоматически обработать ошибку.

* Все методы в Bot API нечувствительны к регистру.
* Все запросы должны выполняться с использованием UTF-8.

Есть два взаимоисключающих способа получения обновлений для бота - метод [getUpdates](https://core.telegram.org/bots/api" \l "getupdates) с одной стороны и [Webhooks](https://core.telegram.org/bots/api" \l "setwebhook) с другой. Входящие обновления хранятся на сервере до тех пор, пока бот не получит их в любом случае, но они не будут храниться дольше 24 часов.

Метод [getUpdates](https://core.telegram.org/bots/api#getupdates) используется для получения входящих обновлений с помощью длительного опроса. Возвращается массив объектов обновления.

При работе с [getUpdates](https://core.telegram.org/bots/api#getupdates)  cледует учесть, что:

* Этот метод не будет работать, если настроен исходящий веб-перехватчик.
* Во избежание дублирования обновлений следует пересчитывать смещение после каждого ответа сервера.

Листинг программного кода библиотеки предоставлен в Приложение А.

## 3.2.3 API Национального Банка Республики Беларусь

Для работы с курсом валют используется API, предоставляемое Национальным Баноком республики баларусь «API получения официального курса белорусского рубля по отношению к иностранным валютам, устанавливаемого Национальным банком Республики Беларусь».

Запрос имеет вид: [https://www.nbrb.by/api/exrates/currencies[/%7Bcur\_id%7D](https://www.nbrb.by/api/exrates/currencies%5b/%7Bcur_id%7D)]

Результат: Возвращает массив объектов класса Currency. Если указан cur\_id, то возвращается один объект Currency.

Класс Currency содержит следующие свойства:

* Cur\_ID – внутренний код
* Cur\_ParentID – этот код используется для связи, при изменениях наименования, количества единиц к которому устанавливается курс белорусского рубля, буквенного, цифрового кодов и т.д. фактически одной и той же валюты\*.
* Cur\_Code – цифровой код
* Cur\_Abbreviation – буквенный код
* Cur\_Name – наименование валюты на русском языке
* Cur\_Name\_Bel – наименование на белорусском языке
* Cur\_Name\_Eng – наименование на английском языке
* Cur\_QuotName – наименование валюты на русском языке, содержащее количество единиц
* Cur\_QuotName\_Bel – наименование на белорусском языке, содержащее количество единиц
* Cur\_QuotName\_Eng – наименование на английском языке, содержащее количество единиц
* Cur\_NameMulti – наименование валюты на русском языке во множественном числе
* Cur\_Name\_BelMulti – наименование валюты на белорусском языке во множественном числе\*
* Cur\_Name\_EngMulti – наименование на английском языке во множественном числе\*
* Cur\_Scale – количество единиц иностранной валюты
* Cur\_Periodicity – периодичность установления курса (0 – ежедневно, 1 – ежемесячно)
* Cur\_DateStart – дата включения валюты в перечень валют, к которым устанавливается официальный курс бел. Рубля
* Cur\_DateEnd – дата исключения валюты из перечня валют, к которым устанавливается официальный курс бел. Рубля

Так же API Национального Банка Республики Беларусь позволяет получить официальный курс белорусского рубля по отношению к иностранным валютам, устанавливаемый Национальным банком на конкретную дату.

Адрес запроса: [https://www.nbrb.by/api/exrates/rates[/{cur\_id}](https://www.nbrb.by/api/exrates/rates%5b/%7bcur_id%7d)]

Параметры (GET):

* ondate\*\* – дата, на которую запрашивается курс (если не задана, то возвращается курс на сегодня)
* periodicity – периодичность установления курса (0 – ежедневно, 1 – ежемесячно)
* parammode – формат аргумента cur\_id: 0 – внутренний код валюты, 1 – трехзначный цифровой код валюты в соответствии со стандартом ИСО 4217, 2 – трехзначный буквенный код валюты (ИСО 4217). По умолчанию = 0

При использовании буквенного или цифрового кода валюты (ИСО 4217) учитывайте его значение на запрашиваемую дату.

Результат: возвращает массив объектов класса Rate. Если указан cur\_id, товозвращается один объект Rate.

Класс Rate содержит следующие свойства:

* Cur\_ID – внутренний код
* Date – дата, на которую запрашивается курс
* Cur\_Abbreviation – буквенный код
* Cur\_Scale – количество единиц иностранной валюты
* Cur\_Name – наименование валюты на русском языке во множественном, либо в единственном числе, в зависимости от количества единиц
* Cur\_OfficialRate – курс\*

API Национального Банка Республики Беларусь предоставляет динамику официального курса белорусского рубля по отношению к заданной иностранной валюте (не более чем за 365 дней).

Параметры (GET):

* startdate\*\* – дата начала запрашиваемого периода
* enddate\*\* – дата окончания запрашиваемого периода

Результат: возвращает массив объектов класса RateShort.

Класс RateShort содержит следующие свойства:

* Cur\_ID – внутренний код
* Date – дата, на которую запрашивается курс
* Cur\_OfficialRate – курс\*

Примечание. Если в любой из перечисленных методов передается неверный код валюты, то возвращается 404 ошибка. Если на запрашиваемую дату курс не установлен, возвращается пустой массив.

## 3.2.4 Описание основных функций разработанного Web-приложения

*Main.rb*

Файл содержит основной алгоритм работы Telegram Бота.

Данные строки подключают библиотеки для работы с API бота и HTTP/HTTPS:

require 'telegram/bot'

require 'httparty'

Метод Telegram::Bot::Client.run(TOKEN) отслеживает изменения на API Telegram бота и записывает полученные изменения в bot. С помощью bot.listen do |message| происходит перебор полученных данных.

CASE - WHEN предназначен для сравнения результата сообщения, которое поступило к боту от пользователя.

Telegram::Bot::Types::CallbackQuery – отслеживание колбэк запросов.

Telegram::Bot::Types::Message – отслеживание сообщения от пользователя.

Приведённая ниже часть кода производит выдачу ответа на основе тектита, поступающего, содержащегося в колбэк.

when Telegram::Bot::Types::CallbackQuery

case message.data

when 'rage'

...

when 'list'

...

when 'help'

...

end

end

Блок кода для обработки команды /start:

kb = [Telegram::Bot::Types::InlineKeyboardButton.new(text: 'Курс доллара', callback\_data: 'rage'), Telegram::Bot::Types::InlineKeyboardButton.new(text: 'Доступные коды валют', callback\_data: 'list'), Telegram::Bot::Types::InlineKeyboardButton.new(text: 'Помощь', callback\_data: 'help')]

markup = Telegram::Bot::Types::InlineKeyboardMarkup.new(inline\_keyboard: kb)

bot.api.send\_message(

chat\_id: message.chat.id,

text: "Добро пожаловать, #{message.from.first\_name}",

reply\_markup: markup

)

Пользователь получает сообщения приветствия и 3 кнопки: «Курс доллара», «Доступные коды валют» и «Помощь».

Блок кода для обработки команды /rage. Следует учесть, что бот поддерживает команду /rage в 2 вариациях:

* /rage КОД\_ВАЛЮТЫ – курс относительно банковской единицы запрашиваемой валюты.
* /rage КОД\_ВАЛЮТЫ СУММЫ – курс относительно указанной суммы запрашиваемой валюты.

Для обработки курса относительно банковской единицы запрашиваемой валюты используется блок кода:

text = "#{response["Cur\_Scale"].to\_f.round(3)} #{response["Cur\_Abbreviation"]} = #{response["Cur\_OfficialRate"].to\_f.round(3)} BYR "

bot.api.send\_message(

chat\_id: message.chat.id,

text: text

)

Для обработки курса относительно указанной суммы запрашиваемой валюты используется блок кода:

count = msg\_arr[2].to\_f

req\_currency = (response["Cur\_Scale"].to\_f / response["Cur\_Scale"].to\_f) \* count

res\_currency = (response["Cur\_OfficialRate"].to\_f / response["Cur\_Scale"].to\_f) \* count

text = "#{ req\_currency.to\_f.round(3) } #{response["Cur\_Abbreviation"]} = #{ res\_currency.to\_f.round(3) } BYR "

bot.api.send\_message(

chat\_id: message.chat.id,

text: text

)

Блок кода для обработки команды /help:

text = "/rage CODE - Получение курса валюты по ее буквенному коду (CODE).\n/rage CODE SUM - Получение курса валюты по ее буквенному коду (CODE) и количеству (SUM)."

bot.api.send\_message(

chat\_id: message.chat.id,

text: text

)

Пользователь получает информацию о доступных командах Telegram Bot.

Блок кода для обработки команды /list:

url = "https://www.nbrb.by/api/exrates/currencies"

response = HTTParty.get(url)

codies = ""

for code in response

if !codies.include? code["Cur\_Abbreviation"]

codies += "#{code["Cur\_Abbreviation"]} - #{code["Cur\_Name"]}\n"

end

end

bot.api.send\_message(

chat\_id: message.chat.id,

text: codies

)

Пользователь получает информацию о доступных кодах валют и их названиях.

При написании кода применялось негласное правило «Золотой код», стандарт W3C и протокол конфиденциальности обмена данными между ресурсом и пользовательским устройством HTTPS.

# ЗАКЛЮЧЕНИЕ

ТОДУ

Сегодня каждый крупный разработчик должен помнить, что, забывая про тестирование программного обеспечения, или же, просто не обращая на него внимания, он фактически обрекает свой продукт на убытки. Ошибки всегда будут присутствовать в любом продукте, ведь человек не идеален. Но с помощью тестирования большую их часть можно избежать. Именно поэтому, тестирование и позволяет разработчикам быть уверенными в собственном программном обеспечении.

В процессе выполнения данного курсового проекта были изучены теоретические сведения в области тестирования программного обеспечения, было разработано и протестировано Web-приложение.

Перед приступлением к практической части были изучены теоретические сведения. Я ознакомился с понятием тестирования, его основными методами, а также с актуальностю его проведения в современном мире.

В процессе разработки Web-приложения были приобретены навыки по работе с такими языками программирования, как: PHP, CSS, HTML и JavaScript. При разработке Web-приложения все поставленные цели, такие как: создание возможности добавления пользователя, создание списка группы и авторизации были выполнены. Также были выполнены основные поставленные задачи.

Было выполнено тестирование разработанного Web-приложения. Тестирование было произведено в ручную, а его результаты были занесены в соответсвующую таблицу.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Экосистема Ruby (on Rails) с горьким привкусом [Электронный ресурс] — Режим доступа: <https://habr.com/ru/post/306564/> — Дата доступа: 06.11.2020.
2. HTTPS Requests in Ruby [Электронный ресурс] — Режим доступа: <https://stackoverflow.com/questions/12883650/https-requests-in-ruby> — Дата доступа: 06.11.2020.
3. RubyGems Digest 2020·09 [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/522648/ — Дата доступа: 06.11.2020.
4. TelegramAPIs [Электронный ресурс]. – Режим доступа: <https://core.telegram.org/#tdlib-build-your-own-telegram> — Дата доступа: 06.11.2020.

# Приложение А – Листинг программного кода

*Файл bot.rb*

require 'virtus'

require 'inflecto'

require 'logger'

require 'json'

require 'faraday'

require 'telegram/bot/types'

require 'telegram/bot/exceptions'

require 'telegram/bot/api'

require 'telegram/bot/null\_logger'

require 'telegram/bot/client'

require 'telegram/bot/version'

require 'telegram/bot/configuration'

module Telegram

module Bot

class << self

attr\_writer :configuration

end

def self.configuration

@configuration ||= Configuration.new

end

def self.configure

yield(configuration)

end

end

end

*Файл api.rb*

module Telegram

module Bot

class Api # rubocop:disable ClassLength

ENDPOINTS = %w(

getUpdates setWebhook deleteWebhook getWebhookInfo getMe sendMessage

forwardMessage sendPhoto sendAudio sendDocument sendVideo sendVoice

sendVideoNote sendMediaGroup sendLocation editMessageLiveLocation

stopMessageLiveLocation sendVenue sendContact sendChatAction

getUserProfilePhotos getFile kickChatMember unbanChatMember

restrictChatMember promoteChatMember leaveChat getChat

getChatAdministrators exportChatInviteLink setChatPhoto deleteChatPhoto

setChatTitle setChatDescription pinChatMessage unpinChatMessage

getChatMembersCount getChatMember setChatStickerSet deleteChatStickerSet

answerCallbackQuery editMessageText editMessageCaption

editMessageReplyMarkup deleteMessage sendSticker getStickerSet

uploadStickerFile createNewStickerSet addStickerToSet

setStickerPositionInSet deleteStickerFromSet answerInlineQuery

sendInvoice answerShippingQuery answerPreCheckoutQuery

sendGame setGameScore getGameHighScores setPassportDataErrors

editMessageMedia sendAnimation sendPoll stopPoll setChatPermissions

setChatAdministratorCustomTitle sendDice getMyCommands setMyCommands

setStickerSetThumb

).freeze

REPLY\_MARKUP\_TYPES = [

Telegram::Bot::Types::ReplyKeyboardMarkup,

Telegram::Bot::Types::ReplyKeyboardRemove,

Telegram::Bot::Types::ForceReply,

Telegram::Bot::Types::InlineKeyboardMarkup

].freeze

INLINE\_QUERY\_RESULT\_TYPES = [

Telegram::Bot::Types::InlineQueryResultArticle,

Telegram::Bot::Types::InlineQueryResultPhoto,

Telegram::Bot::Types::InlineQueryResultGif,

Telegram::Bot::Types::InlineQueryResultMpeg4Gif,

Telegram::Bot::Types::InlineQueryResultVideo,

Telegram::Bot::Types::InlineQueryResultAudio,

Telegram::Bot::Types::InlineQueryResultVoice,

Telegram::Bot::Types::InlineQueryResultDocument,

Telegram::Bot::Types::InlineQueryResultLocation,

Telegram::Bot::Types::InlineQueryResultVenue,

Telegram::Bot::Types::InlineQueryResultContact,

Telegram::Bot::Types::InlineQueryResultGame,

Telegram::Bot::Types::InlineQueryResultCachedPhoto,

Telegram::Bot::Types::InlineQueryResultCachedGif,

Telegram::Bot::Types::InlineQueryResultCachedMpeg4Gif,

Telegram::Bot::Types::InlineQueryResultCachedSticker,

Telegram::Bot::Types::InlineQueryResultCachedDocument,

Telegram::Bot::Types::InlineQueryResultCachedVideo,

Telegram::Bot::Types::InlineQueryResultCachedVoice,

Telegram::Bot::Types::InlineQueryResultCachedAudio

].freeze

attr\_reader :token, :url

def initialize(token, url: 'https://api.telegram.org')

@token = token

@url = url

end

def method\_missing(method\_name, \*args, &block)

endpoint = method\_name.to\_s

endpoint = camelize(endpoint) if endpoint.include?('\_')

ENDPOINTS.include?(endpoint) ? call(endpoint, \*args) : super

end

def respond\_to\_missing?(\*args)

method\_name = args[0].to\_s

method\_name = camelize(method\_name) if method\_name.include?('\_')

ENDPOINTS.include?(method\_name) || super

end

def call(endpoint, raw\_params = {})

params = build\_params(raw\_params)

response = conn.post("/bot#{token}/#{endpoint}", params)

if response.status == 200

JSON.parse(response.body)

else

raise Exceptions::ResponseError.new(response),

'Telegram API has returned the error.'

end

end

private

def build\_params(h)

h.each\_with\_object({}) do |(key, value), params|

params[key] = sanitize\_value(value)

end

end

def sanitize\_value(value)

jsonify\_inline\_query\_results(jsonify\_reply\_markup(value))

end

def jsonify\_reply\_markup(value)

return value unless REPLY\_MARKUP\_TYPES.any? { |type| value.is\_a?(type) }

value.to\_compact\_hash.to\_json

end

def jsonify\_inline\_query\_results(value)

return value unless

value.is\_a?(Array) &&

value.all? { |i| INLINE\_QUERY\_RESULT\_TYPES.any? { |t| i.is\_a?(t) } }

value.map { |i| i.to\_compact\_hash.select { |\_, v| v } }.to\_json

end

def camelize(method\_name)

words = method\_name.split('\_')

words.drop(1).map(&:capitalize!)

words.join

end

def conn

@conn ||= Faraday.new(url: url) do |faraday|

faraday.request :multipart

faraday.request :url\_encoded

faraday.adapter Telegram::Bot.configuration.adapter

end

end

end

end

end

*Файл client.rb*

module Telegram

module Bot

class Client

attr\_reader :api, :options

attr\_accessor :logger

def self.run(\*args, &block)

new(\*args).run(&block)

end

def initialize(token, hash = {})

@options = default\_options.merge(hash)

@api = Api.new(token, url: options.delete(:url))

@logger = options.delete(:logger)

end

def run

yield self

end

def listen(&block)

logger.info('Starting bot')

running = true

Signal.trap('INT') { running = false }

fetch\_updates(&block) while running

exit

end

def fetch\_updates

response = api.getUpdates(options)

return unless response['ok']

response['result'].each do |data|

update = Types::Update.new(data)

@options[:offset] = update.update\_id.next

message = update.current\_message

log\_incoming\_message(message)

yield message

end

rescue Faraday::TimeoutError

retry

end

private

def default\_options

{

offset: 0,

timeout: 20,

logger: NullLogger.new,

url: 'https://api.telegram.org'

}

end

def log\_incoming\_message(message)

uid = message.respond\_to?(:from) && message.from ? message.from.id : nil

logger.info(

format('Incoming message: text="%s" uid=%s', message, uid)

)

end

end

end

end

*Файл types.rb*

require 'telegram/bot/types/compactable'

require 'telegram/bot/types/base'

require 'telegram/bot/types/user'

require 'telegram/bot/types/photo\_size'

require 'telegram/bot/types/audio'

require 'telegram/bot/types/document'

require 'telegram/bot/types/mask\_position'

require 'telegram/bot/types/sticker'

require 'telegram/bot/types/sticker\_set'

require 'telegram/bot/types/video'

require 'telegram/bot/types/voice'

require 'telegram/bot/types/video\_note'

require 'telegram/bot/types/contact'

require 'telegram/bot/types/dice'

require 'telegram/bot/types/location'

require 'telegram/bot/types/chat\_photo'

require 'telegram/bot/types/chat\_permissions'

require 'telegram/bot/types/chat'

require 'telegram/bot/types/message\_entity'

require 'telegram/bot/types/venue'

require 'telegram/bot/types/animation'

require 'telegram/bot/types/game'

require 'telegram/bot/types/callback\_game'

require 'telegram/bot/types/game\_high\_score'

require 'telegram/bot/types/invoice'

require 'telegram/bot/types/shipping\_address'

require 'telegram/bot/types/order\_info'

require 'telegram/bot/types/successful\_payment'

require 'telegram/bot/types/poll\_option'

require 'telegram/bot/types/poll\_answer'

require 'telegram/bot/types/poll'

require 'telegram/bot/types/passport\_file'

require 'telegram/bot/types/encrypted\_passport\_element'

require 'telegram/bot/types/encrypted\_credentials'

require 'telegram/bot/types/passport\_data'

require 'telegram/bot/types/passport\_element\_error\_data\_field'

require 'telegram/bot/types/passport\_element\_error\_front\_side'

require 'telegram/bot/types/passport\_element\_error\_reverse\_side'

require 'telegram/bot/types/passport\_element\_error\_selfie'

require 'telegram/bot/types/passport\_element\_error\_file'

require 'telegram/bot/types/passport\_element\_error\_files'

require 'telegram/bot/types/passport\_element\_error\_translation\_file'

require 'telegram/bot/types/passport\_element\_error\_translation\_files'

require 'telegram/bot/types/passport\_element\_error\_unspecified'

require 'telegram/bot/types/input\_message\_content'

require 'telegram/bot/types/input\_contact\_message\_content'

require 'telegram/bot/types/input\_location\_message\_content'

require 'telegram/bot/types/input\_text\_message\_content'

require 'telegram/bot/types/input\_venue\_message\_content'

require 'telegram/bot/types/login\_url'

require 'telegram/bot/types/inline\_keyboard\_button'

require 'telegram/bot/types/inline\_keyboard\_markup'

require 'telegram/bot/types/inline\_query'

require 'telegram/bot/types/inline\_query\_result\_article'

require 'telegram/bot/types/inline\_query\_result\_audio'

require 'telegram/bot/types/inline\_query\_result\_cached\_audio'

require 'telegram/bot/types/inline\_query\_result\_cached\_document'

require 'telegram/bot/types/inline\_query\_result\_cached\_gif'

require 'telegram/bot/types/inline\_query\_result\_cached\_mpeg4\_gif'

require 'telegram/bot/types/inline\_query\_result\_cached\_photo'

require 'telegram/bot/types/inline\_query\_result\_cached\_sticker'

require 'telegram/bot/types/inline\_query\_result\_cached\_video'

require 'telegram/bot/types/inline\_query\_result\_cached\_voice'

require 'telegram/bot/types/inline\_query\_result\_contact'

require 'telegram/bot/types/inline\_query\_result\_game'

require 'telegram/bot/types/inline\_query\_result\_document'

require 'telegram/bot/types/inline\_query\_result\_gif'

require 'telegram/bot/types/inline\_query\_result\_location'

require 'telegram/bot/types/inline\_query\_result\_mpeg4\_gif'

require 'telegram/bot/types/inline\_query\_result\_photo'

require 'telegram/bot/types/inline\_query\_result\_venue'

require 'telegram/bot/types/inline\_query\_result\_video'

require 'telegram/bot/types/inline\_query\_result\_voice'

require 'telegram/bot/types/chosen\_inline\_result'

require 'telegram/bot/types/message'

require 'telegram/bot/types/callback\_query'

require 'telegram/bot/types/shipping\_query'

require 'telegram/bot/types/pre\_checkout\_query'

require 'telegram/bot/types/update'

require 'telegram/bot/types/keyboard\_button\_poll\_type'

require 'telegram/bot/types/keyboard\_button'

require 'telegram/bot/types/reply\_keyboard\_markup'

require 'telegram/bot/types/reply\_keyboard\_remove'

require 'telegram/bot/types/force\_reply'

require 'telegram/bot/types/file'

require 'telegram/bot/types/labeled\_price'

require 'telegram/bot/types/shipping\_option'

require 'telegram/bot/types/chat\_member'

require 'telegram/bot/types/user\_profile\_photos'

require 'telegram/bot/types/input\_media\_photo'

require 'telegram/bot/types/input\_media\_video'

require 'telegram/bot/types/input\_media\_animation'

require 'telegram/bot/types/input\_media\_audio'

require 'telegram/bot/types/input\_media\_document'

Virtus.finalize

*Файл response\_error.rb*

module Telegram

module Bot

module Exceptions

class ResponseError < Base

attr\_reader :response

def initialize(response)

@response = response

end

def to\_s

super +

format(' (%s)', data.map { |k, v| %(#{k}: "#{v}") }.join(', '))

end

def error\_code

data[:error\_code] || data['error\_code']

end

private

def data

@data ||= begin

JSON.parse(response.body)

rescue JSON::ParserError

{ error\_code: response.status, uri: response.env.url.to\_s }

end

end

end

end

end

end

*Файл api\_spec.rb*

RSpec.describe Telegram::Bot::Api do

let(:token) { '180956132:AAHU0\_CeyQWOd6baBc9TibTPybxY9p1P8xo' }

let(:endpoint) { 'getMe' }

let(:api) { described\_class.new(token) }

describe '#call' do

subject(:api\_call) { api.call(endpoint) }

it 'returns hash' do

is\_expected.to be\_a(Hash)

end

it 'has status' do

is\_expected.to have\_key('ok')

end

it 'has result' do

is\_expected.to have\_key('result')

end

context 'when token is invalid' do

let(:token) { '123456:wrongtoken' }

it 'raises an error' do

expect { api\_call }

.to raise\_error(Telegram::Bot::Exceptions::ResponseError)

end

end

end

describe '#method\_missing' do

subject { api }

it 'responds to endpoints' do

is\_expected.to respond\_to(endpoint)

end

context 'when method name is in snake case' do

let(:endpoint) { 'get\_me' }

it 'responds to snake-cased endpoints' do

is\_expected.to respond\_to(endpoint)

end

end

end

end

*Файл client\_spec.rb*

RSpec.describe Telegram::Bot::Client do

let(:token) { '180956132:AAHU0\_CeyQWOd6baBc9TibTPybxY9p1P8xo' }

let(:client) { described\_class.new(token, options) }

describe '#new' do

subject { client.options }

context 'with any options' do

let(:options) do

{

offset: 1,

timeout: 30,

something: true

}

end

it 'accepts any option' do

is\_expected.to eql(options)

end

end

context 'with logger option' do

let(:options) do

{

offset: 1,

timeout: 30,

something: true,

logger: Logger.new('/dev/null')

}

end

it 'removes logger from options' do

is\_expected.not\_to include(:logger)

end

end

end

end

*Файл telegram-bot-ruby.gemspec*

lib = File.expand\_path('../lib', \_\_FILE\_\_)

$LOAD\_PATH.unshift(lib) unless $LOAD\_PATH.include?(lib)

require 'telegram/bot/version'

Gem::Specification.new do |spec|

spec.name = 'telegram-bot-ruby'

spec.version = Telegram::Bot::VERSION

spec.authors = ['Alexander Tipugin']

spec.email = ['atipugin@gmail.com']

spec.summary = "Ruby wrapper for Telegram's Bot API"

spec.homepage = 'https://github.com/atipugin/telegram-bot'

spec.files = `git ls-files -z`.split("\x0").reject { |f| f.match(%r{^(test|spec|features)/}) }

spec.bindir = 'exe'

spec.executables = spec.files.grep(%r{^exe/}) { |f| File.basename(f) }

spec.require\_paths = ['lib']

spec.add\_dependency 'faraday'

spec.add\_dependency 'virtus'

spec.add\_dependency 'inflecto'

spec.add\_development\_dependency 'rake', '~> 13.0'

spec.add\_development\_dependency 'pry'

spec.add\_development\_dependency 'rubocop', '~> 0.48.1'

spec.add\_development\_dependency 'rspec', '~> 3.4'

spec.add\_development\_dependency 'rubocop-rspec', '~> 1.8'

end

*Файл spec*

set -e

cd $(dirname "$0")/..

[ "$1" = "--help" -o "$1" = "-h" -o "$1" = "help" ] && {

grep '^#/' <"$0"| cut -c4-

exit 0

}

gem\_name=httparty

# Build a new gem archive.

rm -rf $gem\_name-\*.gem

gem build -q $gem\_name.gemspec

# Make sure we're on the master branch.

(git branch | grep -q '\* master') || {

echo "Only release from the master branch."

exit 1

}

# Figure out what version we're releasing.

tag=v`ls $gem\_name-\*.gem | sed "s/^$gem\_name-\(.\*\)\.gem$/\1/"`

echo "Releasing $tag"

# Make sure we haven't released this version before.

git fetch -t origin

(git tag -l | grep -q "$tag") && {

echo "Whoops, there's already a '${tag}' tag."

exit 1

}

# Tag it and bag it.

gem push $gem\_name-\*.gem && git tag "$tag" &&

git push origin master && git push origin "$tag"

*Файл httparty*

require\_relative 'spec\_helper'

RSpec.describe HTTParty do

before(:each) do

@klass = Class.new

@klass.instance\_eval { include HTTParty }

end

describe "pem" do

it 'should set the pem content' do

@klass.pem 'PEM-CONTENT'

expect(@klass.default\_options[:pem]).to eq('PEM-CONTENT')

end

it "should set the password to nil if it's not provided" do

@klass.pem 'PEM-CONTENT'

expect(@klass.default\_options[:pem\_password]).to be\_nil

end

it 'should set the password' do

@klass.pem 'PEM-CONTENT', 'PASSWORD'

expect(@klass.default\_options[:pem\_password]).to eq('PASSWORD')

end

end

describe "pkcs12" do

it 'should set the p12 content' do

@klass.pkcs12 'P12-CONTENT', 'PASSWORD'

expect(@klass.default\_options[:p12]).to eq('P12-CONTENT')

end

it 'should set the password' do

@klass.pkcs12 'P12-CONTENT', 'PASSWORD'

expect(@klass.default\_options[:p12\_password]).to eq('PASSWORD')

end

end

describe 'ssl\_version' do

it 'should set the ssl\_version content' do

@klass.ssl\_version :SSLv3

expect(@klass.default\_options[:ssl\_version]).to eq(:SSLv3)

end

end

describe 'ciphers' do

it 'should set the ciphers content' do

expect(@klass.default\_options[:ciphers]).to be\_nil

@klass.ciphers 'RC4-SHA'

expect(@klass.default\_options[:ciphers]).to eq('RC4-SHA')

end

end

describe 'http\_proxy' do

it 'should set the address' do

@klass.http\_proxy 'proxy.foo.com', 80

options = @klass.default\_options

expect(options[:http\_proxyaddr]).to eq('proxy.foo.com')

expect(options[:http\_proxyport]).to eq(80)

end

it 'should set the proxy user and pass when they are provided' do

@klass.http\_proxy 'proxy.foo.com', 80, 'user', 'pass'

options = @klass.default\_options

expect(options[:http\_proxyuser]).to eq('user')

expect(options[:http\_proxypass]).to eq('pass')

end

end

describe "base uri" do

before(:each) do

@klass.base\_uri('api.foo.com/v1')

end

it "should have reader" do

expect(@klass.base\_uri).to eq('http://api.foo.com/v1')

end

it 'should have writer' do

@klass.base\_uri('http://api.foobar.com')

expect(@klass.base\_uri).to eq('http://api.foobar.com')

end

it 'should not modify the parameter during assignment' do

uri = 'http://api.foobar.com'

@klass.base\_uri(uri)

expect(uri).to eq('http://api.foobar.com')

end

end

describe ".disable\_rails\_query\_string\_format" do

it "sets the query string normalizer to HTTParty::Request::NON\_RAILS\_QUERY\_STRING\_NORMALIZER" do

@klass.disable\_rails\_query\_string\_format

expect(@klass.default\_options[:query\_string\_normalizer]).to eq(HTTParty::Request::NON\_RAILS\_QUERY\_STRING\_NORMALIZER)

end

end

describe ".normalize\_base\_uri" do

it "should add http if not present for non ssl requests" do

uri = HTTParty.normalize\_base\_uri('api.foobar.com')

expect(uri).to eq('http://api.foobar.com')

end

it "should add https if not present for ssl requests" do

uri = HTTParty.normalize\_base\_uri('api.foo.com/v1:443')

expect(uri).to eq('https://api.foo.com/v1:443')

end

it "should not remove https for ssl requests" do

uri = HTTParty.normalize\_base\_uri('https://api.foo.com/v1:443')

expect(uri).to eq('https://api.foo.com/v1:443')

end

it 'should not modify the parameter' do

uri = 'http://api.foobar.com'

HTTParty.normalize\_base\_uri(uri)

expect(uri).to eq('http://api.foobar.com')

end

it "should not treat uri's with a port of 4430 as ssl" do

uri = HTTParty.normalize\_base\_uri('http://api.foo.com:4430/v1')

expect(uri).to eq('http://api.foo.com:4430/v1')

end

end

describe "headers" do

def expect\_headers(header = {})

expect(HTTParty::Request).to receive(:new) \

.with(anything, anything, hash\_including({ headers: header })) \

.and\_return(double("mock response", perform: nil))

end

it "does not modify default\_options when no arguments are passed" do

@klass.headers

expect(@klass.default\_options[:headers]).to eq(nil)

end

it "should default to empty hash" do

expect(@klass.headers).to eq({})

end

it "should be able to be updated" do

init\_headers = {foo: 'bar', baz: 'spax'}

@klass.headers init\_headers

expect(@klass.headers).to eq(init\_headers)

end

it "should be able to accept block as header value" do

init\_headers = {'foo' => lambda {'bar'}}

@klass.headers init\_headers

stub\_request(:get, "http://example.com/").with(headers: {'foo' => 'bar', 'baz' => 'spax'})

@klass.get('http://example.com/', headers: {'baz' => -> {'spax'}})

expect(@klass.headers).to eq(init\_headers)

end

it "should pass options as argument to header block value" do

init\_headers = { 'foo' => lambda { |options| options[:body] } }

@klass.headers init\_headers

stub\_request(:get, "http://example.com/").with(body: 'bar', headers: { 'foo' => 'bar', 'baz' => 'rab' })

@klass.get('http://example.com/', body: 'bar',headers: { 'baz' => -> (options){ options[:body].reverse } })

expect(@klass.headers).to eq(init\_headers)

end

it "uses the class headers when sending a request" do

expect\_headers('foo' => 'bar')

@klass.headers(foo: 'bar')

@klass.get('')

end

it "merges class headers with request headers" do

expect\_headers('baz' => 'spax', 'foo' => 'bar')

@klass.headers(foo: 'bar')

@klass.get('', headers: {baz: 'spax'})

end

it 'overrides class headers with request headers' do

expect\_headers('baz' => 'spax', 'foo' => 'baz')

@klass.headers(foo: 'bar')

@klass.get('', headers: {baz: 'spax', foo: 'baz'})

end

context "with cookies" do

it 'utilizes the class-level cookies' do

expect\_headers('foo' => 'bar', 'cookie' => 'type=snickerdoodle')

@klass.headers(foo: 'bar')

@klass.cookies(type: 'snickerdoodle')

@klass.get('')

end

it 'adds cookies to the headers' do

expect\_headers('foo' => 'bar', 'cookie' => 'type=snickerdoodle')

@klass.headers(foo: 'bar')

@klass.get('', cookies: {type: 'snickerdoodle'})

end

it 'doesnt modify default headers' do

expect(@klass.headers).to eq({})

expect\_headers('cookie' => 'type=snickerdoodle')

@klass.get('', cookies: {type: 'snickerdoodle'})

expect(@klass.headers).to eq({})

end

it 'adds optional cookies to the optional headers' do

expect\_headers('baz' => 'spax', 'cookie' => 'type=snickerdoodle')

@klass.get('', cookies: {type: 'snickerdoodle'}, headers: {baz: 'spax'})

end

end

context 'when posting file' do

let(:boundary) { '------------------------c772861a5109d5ef' }

let(:headers) do

{ 'Content-Type'=>"multipart/form-data; boundary=#{boundary}" }

end

before do

expect(HTTParty::Request::MultipartBoundary).to receive(:generate).and\_return(boundary)

end

it 'changes content-type headers to multipart/form-data' do

stub\_request(:post, "http://example.com/").with(headers: headers)

@klass.post('http://example.com', body: { file: File.open('spec/fixtures/tiny.gif')})

end

end

context 'when headers passed as symbols' do

it 'converts them to string' do

expect\_headers('foo' => 'application/json', 'bar' => 'example')

headers = { foo: 'application/json', bar: 'example' }

@klass.post('http://example.com', headers: headers)

end

it 'converts default headers to string' do

expect\_headers('foo' => 'application/json', 'bar' => 'example')

@klass.headers(foo: 'application/json')

@klass.post('http://example.com', headers: { bar: 'example' })

end

end

end

describe "cookies" do

def expect\_cookie\_header(s)

expect(HTTParty::Request).to receive(:new) \

.with(anything, anything, hash\_including({ headers: { "cookie" => s } })) \

.and\_return(double("mock response", perform: nil))

end

it "should not be in the headers by default" do

allow(HTTParty::Request).to receive(:new).and\_return(double(nil, perform: nil))

@klass.get("")

expect(@klass.headers.keys).not\_to include("cookie")

end

it "should raise an ArgumentError if passed a non-Hash" do

expect do

@klass.cookies("nonsense")

end.to raise\_error(ArgumentError)

end

it "should allow a cookie to be specified with a one-off request" do

expect\_cookie\_header "type=snickerdoodle"

@klass.get("", cookies: { type: "snickerdoodle" })

end

describe "when a cookie is set at the class level" do

before(:each) do

@klass.cookies({ type: "snickerdoodle" })

end

it "should include that cookie in the request" do

expect\_cookie\_header "type=snickerdoodle"

@klass.get("")

end

it "should pass the proper cookies when requested multiple times" do

2.times do

expect\_cookie\_header "type=snickerdoodle"

@klass.get("")

end

end

it "should allow the class defaults to be overridden" do

expect\_cookie\_header "type=chocolate\_chip"

@klass.get("", cookies: { type: "chocolate\_chip" })

end

end

describe "in a class with multiple methods that use different cookies" do

before(:each) do

@klass.instance\_eval do

def first\_method

get("first\_method", cookies: { first\_method\_cookie: "foo" })

end

def second\_method

get("second\_method", cookies: { second\_method\_cookie: "foo" })

end

end

end

it "should not allow cookies used in one method to carry over into other methods" do

expect\_cookie\_header "first\_method\_cookie=foo"

@klass.first\_method

expect\_cookie\_header "second\_method\_cookie=foo"

@klass.second\_method

end

end

end

describe "default params" do

it "should default to empty hash" do

expect(@klass.default\_params).to eq({})

end

it "should be able to be updated" do

new\_defaults = {foo: 'bar', baz: 'spax'}

@klass.default\_params new\_defaults

expect(@klass.default\_params).to eq(new\_defaults)

end

end

describe "default timeout" do

it "should default to nil" do

expect(@klass.default\_options[:timeout]).to eq(nil)

end

it "should support updating" do

@klass.default\_timeout 10

expect(@klass.default\_options[:timeout]).to eq(10)

end

it "should support floats" do

@klass.default\_timeout 0.5

expect(@klass.default\_options[:timeout]).to eq(0.5)

end

it "should raise an exception if unsupported type provided" do

expect { @klass.default\_timeout "0.5" }.to raise\_error ArgumentError

end

end

describe "debug\_output" do

it "stores the given stream as a default\_option" do

@klass.debug\_output $stdout

expect(@klass.default\_options[:debug\_output]).to eq($stdout)

end

it "stores the $stderr stream by default" do

@klass.debug\_output

expect(@klass.default\_options[:debug\_output]).to eq($stderr)

end

end

describe "basic http authentication" do

it "should work" do

@klass.basic\_auth 'foobar', 'secret'

expect(@klass.default\_options[:basic\_auth]).to eq({username: 'foobar', password: 'secret'})

end

end

describe "digest http authentication" do

it "should work" do

@klass.digest\_auth 'foobar', 'secret'

expect(@klass.default\_options[:digest\_auth]).to eq({username: 'foobar', password: 'secret'})

end

end

describe "parser" do

class CustomParser

def self.parse(body)

{sexy: true}

end

end

let(:parser) do

proc { |data, format| CustomParser.parse(data) }

end

it "should set parser options" do

@klass.parser parser

expect(@klass.default\_options[:parser]).to eq(parser)

end

it "should be able parse response with custom parser" do

@klass.parser parser

stub\_request(:get, 'http://twitter.com/statuses/public\_timeline.xml')

.to\_return(body: 'tweets')

custom\_parsed\_response = @klass.get('http://twitter.com/statuses/public\_timeline.xml')

expect(custom\_parsed\_response[:sexy]).to eq(true)

end

it "raises UnsupportedFormat when the parser cannot handle the format" do

@klass.format :json

parser\_class = Class.new(HTTParty::Parser)

parser\_class::SupportedFormats = {}

expect do

@klass.parser parser\_class

end.to raise\_error(HTTParty::UnsupportedFormat)

end

it 'does not validate format whe custom parser is a proc' do

expect do

@klass.format :json

@klass.parser lambda {|body, format|}

end.to\_not raise\_error

end

end

describe "uri\_adapter" do

context 'with Addressable::URI' do

before do

@klass.uri\_adapter(Addressable::URI)

end

it 'handles international domains' do

uri = 'http://xn--i-7iqv272g.ws/'

stub\_request(:get, uri).to\_return(body: 'stuff')

response = @klass.get('http://i❤️.ws')

expect(response.parsed\_response).to eq('stuff')

expect(response.request.uri.to\_s).to eq(uri)

end

end

context 'with custom URI Adaptor' do

require 'forwardable'

class CustomURIAdaptor

extend Forwardable

def\_delegators :@uri, :userinfo, :relative?, :query, :query=, :scheme, :path, :host, :port

def initialize(uri)

@uri = uri

end

def self.parse(uri)

new(URI.parse(uri))

end

def normalize

self

end

end

let(:uri\_adapter) { CustomURIAdaptor }

it "should set the uri\_adapter" do

@klass.uri\_adapter uri\_adapter

expect(@klass.default\_options[:uri\_adapter]).to be uri\_adapter

end

it "should raise an ArgumentError if uri\_adapter doesn't implement parse method" do

expect do

@klass.uri\_adapter double()

end.to raise\_error(ArgumentError)

end

it "should process a request with a uri instance parsed from the uri\_adapter" do

uri = 'http://foo.com/bar'

stub\_request(:get, uri).to\_return(body: 'stuff')

@klass.uri\_adapter uri\_adapter

expect(@klass.get(uri).parsed\_response).to eq('stuff')

end

end

end

describe "connection\_adapter" do

let(:uri) { 'http://google.com/api.json' }

let(:connection\_adapter) { double('CustomConnectionAdapter') }

it "should set the connection\_adapter" do

@klass.connection\_adapter connection\_adapter

expect(@klass.default\_options[:connection\_adapter]).to be connection\_adapter

end

it "should set the connection\_adapter\_options when provided" do

options = {foo: :bar}

@klass.connection\_adapter connection\_adapter, options

expect(@klass.default\_options[:connection\_adapter\_options]).to be options

end

it "should not set the connection\_adapter\_options when not provided" do

@klass.connection\_adapter connection\_adapter

expect(@klass.default\_options[:connection\_adapter\_options]).to be\_nil

end

it "should process a request with a connection from the adapter" do

connection\_adapter\_options = {foo: :bar}

expect(connection\_adapter).to receive(:call) { |u, o|

expect(o[:connection\_adapter\_options]).to eq(connection\_adapter\_options)

HTTParty::ConnectionAdapter.call(u, o)

}.with(URI.parse(uri), kind\_of(Hash))

stub\_request(:get, uri).to\_return(body: 'stuff')

@klass.connection\_adapter connection\_adapter, connection\_adapter\_options

expect(@klass.get(uri).parsed\_response).to eq('stuff')

end

end

describe "format" do

it "should allow xml" do

@klass.format :xml

expect(@klass.default\_options[:format]).to eq(:xml)

end

it "should allow csv" do

@klass.format :csv

expect(@klass.default\_options[:format]).to eq(:csv)

end

it "should allow json" do

@klass.format :json

expect(@klass.default\_options[:format]).to eq(:json)

end

it "should allow plain" do

@klass.format :plain

expect(@klass.default\_options[:format]).to eq(:plain)

end

it 'should not allow funky format' do

expect do

@klass.format :foobar

end.to raise\_error(HTTParty::UnsupportedFormat)

end

it 'should only print each format once with an exception' do

expect do

@klass.format :foobar

end.to raise\_error(HTTParty::UnsupportedFormat, "':foobar' Must be one of: csv, html, json, plain, xml")

end

it 'sets the default parser' do

expect(@klass.default\_options[:parser]).to be\_nil

@klass.format :json

expect(@klass.default\_options[:parser]).to eq(HTTParty::Parser)

end

it 'does not reset parser to the default parser' do

my\_parser = lambda {}

@klass.parser my\_parser

@klass.format :json

expect(@klass.parser).to eq(my\_parser)

end

end

describe "#no\_follow" do

it "sets no\_follow to false by default" do

@klass.no\_follow

expect(@klass.default\_options[:no\_follow]).to be\_falsey

end

it "sets the no\_follow option to true" do

@klass.no\_follow true

expect(@klass.default\_options[:no\_follow]).to be\_truthy

end

end

describe "#maintain\_method\_across\_redirects" do

it "sets maintain\_method\_across\_redirects to true by default" do

@klass.maintain\_method\_across\_redirects

expect(@klass.default\_options[:maintain\_method\_across\_redirects]).to be\_truthy

end

it "sets the maintain\_method\_across\_redirects option to false" do

@klass.maintain\_method\_across\_redirects false

expect(@klass.default\_options[:maintain\_method\_across\_redirects]).to be\_falsey

end

end

describe "#resend\_on\_redirect" do

it "sets resend\_on\_redirect to true by default" do

@klass.resend\_on\_redirect

expect(@klass.default\_options[:resend\_on\_redirect]).to be\_truthy

end

it "sets resend\_on\_redirect option to false" do

@klass.resend\_on\_redirect false

expect(@klass.default\_options[:resend\_on\_redirect]).to be\_falsey

end

end

describe ".follow\_redirects" do

it "sets follow redirects to true by default" do

@klass.follow\_redirects

expect(@klass.default\_options[:follow\_redirects]).to be\_truthy

end

it "sets the follow\_redirects option to false" do

@klass.follow\_redirects false

expect(@klass.default\_options[:follow\_redirects]).to be\_falsey

end

end

describe ".query\_string\_normalizer" do

it "sets the query\_string\_normalizer option" do

normalizer = proc {}

@klass.query\_string\_normalizer normalizer

expect(@klass.default\_options[:query\_string\_normalizer]).to eq(normalizer)

end

end

describe ".raise\_on" do

context 'when parameters is an array' do

it 'sets raise\_on option' do

@klass.raise\_on [500, 404]

expect(@klass.default\_options[:raise\_on]).to contain\_exactly(404, 500)

end

end

context 'when parameters is a fixnum' do

it 'sets raise\_on option' do

@klass.raise\_on 404

expect(@klass.default\_options[:raise\_on]).to contain\_exactly(404)

end

end

end

describe "with explicit override of automatic redirect handling" do

before do

@request = HTTParty::Request.new(Net::HTTP::Get, 'http://api.foo.com/v1', format: :xml, no\_follow: true)

@redirect = stub\_response 'first redirect', 302

@redirect['location'] = 'http://foo.com/bar'

allow(HTTParty::Request).to receive\_messages(new: @request)

end

it "should fail with redirected GET" do

expect do

@error = @klass.get('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected POST" do

expect do

@klass.post('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected PATCH" do

expect do

@klass.patch('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected DELETE" do

expect do

@klass.delete('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected MOVE" do

expect do

@klass.move('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected COPY" do

expect do

@klass.copy('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected PUT" do

expect do

@klass.put('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected HEAD" do

expect do

@klass.head('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected OPTIONS" do

expect do

@klass.options('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

it "should fail with redirected MKCOL" do

expect do

@klass.mkcol('/foo', no\_follow: true)

end.to raise\_error(HTTParty::RedirectionTooDeep) {|e| expect(e.response.body).to eq('first redirect')}

end

end

describe "head requests should follow redirects requesting HEAD only" do

before do

allow(HTTParty::Request).to receive(:new).

and\_return(double("mock response", perform: nil))

end

it "should remain HEAD request across redirects, unless specified otherwise" do

expect(@klass).to receive(:ensure\_method\_maintained\_across\_redirects).with({})

@klass.head('/foo')

end

end

describe "#ensure\_method\_maintained\_across\_redirects" do

it "should set maintain\_method\_across\_redirects option if unspecified" do

options = {}

@klass.send(:ensure\_method\_maintained\_across\_redirects, options)

expect(options[:maintain\_method\_across\_redirects]).to be\_truthy

end

it "should not set maintain\_method\_across\_redirects option if value is present" do

options = { maintain\_method\_across\_redirects: false }

@klass.send(:ensure\_method\_maintained\_across\_redirects, options)

expect(options[:maintain\_method\_across\_redirects]).to be\_falsey

end

end

describe "with multiple class definitions" do

before(:each) do

@klass.instance\_eval do

base\_uri "http://first.com"

default\_params one: 1

end

@additional\_klass = Class.new

@additional\_klass.instance\_eval do

include HTTParty

base\_uri "http://second.com"

default\_params two: 2

end

end

it "should not run over each others options" do

expect(@klass.default\_options).to eq({ base\_uri: 'http://first.com', default\_params: { one: 1 } })

expect(@additional\_klass.default\_options).to eq({ base\_uri: 'http://second.com', default\_params: { two: 2 } })

end

end

describe "two child classes inheriting from one parent" do

before(:each) do

@parent = Class.new do

include HTTParty

def self.name

"Parent"

end

end

@child1 = Class.new(@parent)

@child2 = Class.new(@parent)

end

it "does not modify each others inherited attributes" do

@child1.default\_params joe: "alive"

@child2.default\_params joe: "dead"

expect(@child1.default\_options).to eq({ default\_params: {joe: "alive"} })

expect(@child2.default\_options).to eq({ default\_params: {joe: "dead"} })

expect(@parent.default\_options).to eq({ })

end

it "inherits default\_options from the superclass" do

@parent.basic\_auth 'user', 'password'

expect(@child1.default\_options).to eq({basic\_auth: {username: 'user', password: 'password'}})

@child1.basic\_auth 'u', 'p' # modifying child1 has no effect on child2

expect(@child2.default\_options).to eq({basic\_auth: {username: 'user', password: 'password'}})

end

it "doesn't modify the parent's default options" do

@parent.basic\_auth 'user', 'password'

@child1.basic\_auth 'u', 'p'

expect(@child1.default\_options).to eq({basic\_auth: {username: 'u', password: 'p'}})

@child1.basic\_auth 'email', 'token'

expect(@child1.default\_options).to eq({basic\_auth: {username: 'email', password: 'token'}})

expect(@parent.default\_options).to eq({basic\_auth: {username: 'user', password: 'password'}})

end

it "doesn't modify hashes in the parent's default options" do

@parent.headers 'Accept' => 'application/json'

@child1.headers 'Accept' => 'application/xml'

expect(@parent.default\_options[:headers]).to eq({'Accept' => 'application/json'})

expect(@child1.default\_options[:headers]).to eq({'Accept' => 'application/xml'})

end

it "works with lambda values" do

@child1.default\_options[:imaginary\_option] = lambda { "This is a new lambda "}

expect(@child1.default\_options[:imaginary\_option]).to be\_a Proc

end

it 'should dup the proc on the child class' do

imaginary\_option = lambda { 2 \* 3.14 }

@parent.default\_options[:imaginary\_option] = imaginary\_option

expect(@parent.default\_options[:imaginary\_option].call).to eq(imaginary\_option.call)

@child1.default\_options[:imaginary\_option]

expect(@child1.default\_options[:imaginary\_option].call).to eq(imaginary\_option.call)

expect(@child1.default\_options[:imaginary\_option]).not\_to be\_equal imaginary\_option

end

it "inherits default\_cookies from the parent class" do

@parent.cookies 'type' => 'chocolate\_chip'

expect(@child1.default\_cookies).to eq({"type" => "chocolate\_chip"})

@child1.cookies 'type' => 'snickerdoodle'

expect(@child1.default\_cookies).to eq({"type" => "snickerdoodle"})

expect(@child2.default\_cookies).to eq({"type" => "chocolate\_chip"})

end

it "doesn't modify the parent's default cookies" do

@parent.cookies 'type' => 'chocolate\_chip'

@child1.cookies 'type' => 'snickerdoodle'

expect(@child1.default\_cookies).to eq({"type" => "snickerdoodle"})

expect(@parent.default\_cookies).to eq({"type" => "chocolate\_chip"})

end

end

describe "grand parent with inherited callback" do

before do

@grand\_parent = Class.new do

def self.inherited(subclass)

subclass.instance\_variable\_set(:@grand\_parent, true)

end

end

@parent = Class.new(@grand\_parent) do

include HTTParty

end

end

it "continues running the #inherited on the parent" do

child = Class.new(@parent)

expect(child.instance\_variable\_get(:@grand\_parent)).to be\_truthy

end

end

describe "#get" do

it "should be able to get html" do

stub\_http\_response\_with('example.html')

expect(HTTParty.get('http://www.example.com').parsed\_response).to eq(file\_fixture('example.html'))

end

it "should be able to get chunked html" do

chunks = %w(Chunk1 Chunk2 Chunk3 Chunk4)

stub\_chunked\_http\_response\_with(chunks)

expect(

HTTParty.get('http://www.google.com') do |fragment|

expect(chunks).to include(fragment)

end.parsed\_response

).to eq(chunks.join)

end

it "should return an empty body if stream\_body option is turned on" do

chunks = %w(Chunk1 Chunk2 Chunk3 Chunk4)

options = {stream\_body: true, format: 'html'}

stub\_chunked\_http\_response\_with(chunks, options)

expect(

HTTParty.get('http://www.google.com', options) do |fragment|

expect(chunks).to include(fragment)

expect(fragment.code).to eql 200

expect(fragment.http\_response).to be

end.parsed\_response

).to eq(nil)

end

context 'when streaming body' do

let(:chunk) { 'Content'.force\_encoding('ascii-8bit') }

let(:options) { { stream\_body: true } }

before do

stub\_chunked\_http\_response\_with([chunk], options) do |response|

allow(response).to receive(:[]).with('content-type').and\_return('text/plain; charset = "utf-8"')

end

end

specify do

HTTParty.get('http://www.google.com', options) do |fragment|

expect(fragment.encoding).to eq(Encoding.find("UTF-8"))

end

end

end

it "should be able parse response type json automatically" do

stub\_http\_response\_with('twitter.json')

tweets = HTTParty.get('http://twitter.com/statuses/public\_timeline.json')

expect(tweets.size).to eq(20)

expect(tweets.first['user']).to eq({

"name" => "Pyk",

"url" => nil,

"id" => "7694602",

"description" => nil,

"protected" => false,

"screen\_name" => "Pyk",

"followers\_count" => 1,

"location" => "Opera Plaza, California",

"profile\_image\_url" => "http://static.twitter.com/images/default\_profile\_normal.png"

})

end

it "should be able parse response type xml automatically" do

stub\_http\_response\_with('twitter.xml')

tweets = HTTParty.get('http://twitter.com/statuses/public\_timeline.xml')

expect(tweets['statuses'].size).to eq(20)

expect(tweets['statuses'].first['user']).to eq({

"name" => "Magic 8 Bot",

"url" => nil,

"id" => "17656026",

"description" => "ask me a question",

"protected" => "false",

"screen\_name" => "magic8bot",

"followers\_count" => "90",

"profile\_image\_url" => "http://s3.amazonaws.com/twitter\_production/profile\_images/65565851/8ball\_large\_normal.jpg",

"location" => nil

})

end

it "should be able parse response type csv automatically" do

stub\_http\_response\_with('twitter.csv')

profile = HTTParty.get('http://twitter.com/statuses/profile.csv')

expect(profile.size).to eq(2)

expect(profile[0]).to eq(%w(name url id description protected screen\_name followers\_count profile\_image\_url location))

expect(profile[1]).to eq(["Magic 8 Bot", nil, "17656026", "ask me a question", "false", "magic8bot", "90", "http://s3.amazonaws.com/twitter\_production/profile\_images/65565851/8ball\_large\_normal.jpg", nil])

end

it "should not get undefined method add\_node for nil class for the following xml" do

stub\_http\_response\_with('undefined\_method\_add\_node\_for\_nil.xml')

result = HTTParty.get('http://foobar.com')

expect(result.parsed\_response).to eq({"Entities" => {"href" => "https://s3-sandbox.parature.com/api/v1/5578/5633/Account", "results" => "0", "total" => "0", "page\_size" => "25", "page" => "1"}})

end

it "should parse empty response fine" do

stub\_http\_response\_with('empty.xml')

result = HTTParty.get('http://foobar.com')

expect(result).to be\_nil

end

it "should accept http URIs" do

stub\_http\_response\_with('example.html')

expect do

HTTParty.get('http://example.com')

end.not\_to raise\_error

end

it "should accept https URIs" do

stub\_http\_response\_with('example.html')

expect do

HTTParty.get('https://example.com')

end.not\_to raise\_error

end

it "should accept webcal URIs" do

uri = 'http://google.com/'

stub\_request(:get, uri).to\_return(body: 'stuff')

uri = 'webcal://google.com/'

expect do

HTTParty.get(uri)

end.not\_to raise\_error

end

it "should raise an InvalidURIError on URIs that can't be parsed at all" do

expect do

HTTParty.get("It's the one that says 'Bad URI'")

end.to raise\_error(URI::InvalidURIError)

end

end

end

*Файл client\_helper.rb*

require "simplecov"

SimpleCov.start

require 'pry'

require "httparty"

require 'webmock/rspec'

def file\_fixture(filename)

open(File.join(File.dirname(\_\_FILE\_\_), 'fixtures', "#{filename}")).read

end

Dir[File.expand\_path(File.join(File.dirname(\_\_FILE\_\_), 'support', '\*\*', '\*.rb'))].each {|f| require f}

RSpec.configure do |config|

config.include HTTParty::StubResponse

config.include HTTParty::SSLTestHelper

config.expect\_with :rspec do |expectations|

expectations.include\_chain\_clauses\_in\_custom\_matcher\_descriptions = true

end

config.mock\_with :rspec do |mocks|

mocks.verify\_partial\_doubles = false

end

config.filter\_run :focus

config.run\_all\_when\_everything\_filtered = true

config.disable\_monkey\_patching!

config.warnings = true

if config.files\_to\_run.one?

config.default\_formatter = 'doc'

end

config.profile\_examples = 10

config.order = :random

config.before(:each) do

# Reset default\_cert\_store cache

HTTParty::ConnectionAdapter.instance\_variable\_set(:@default\_cert\_store, nil)

end

Kernel.srand config.seed

end

RSpec::Matchers.define :use\_ssl do

match(&:use\_ssl?)

end

RSpec::Matchers.define :use\_cert\_store do |cert\_store|

match do |connection|

connection.cert\_store == cert\_store

end

end

*Файл connection\_adapter\_spec.rb*

require 'spec\_helper'

RSpec.describe HTTParty::ConnectionAdapter do

describe "initialization" do

let(:uri) { URI 'http://www.google.com' }

it "takes a URI as input" do

HTTParty::ConnectionAdapter.new(uri)

end

it "raises an ArgumentError if the uri is nil" do

expect { HTTParty::ConnectionAdapter.new(nil) }.to raise\_error ArgumentError

end

it "raises an ArgumentError if the uri is a String" do

expect { HTTParty::ConnectionAdapter.new('http://www.google.com') }.to raise\_error ArgumentError

end

it "sets the uri" do

adapter = HTTParty::ConnectionAdapter.new(uri)

expect(adapter.uri).to be uri

end

it "also accepts an optional options hash" do

HTTParty::ConnectionAdapter.new(uri, {})

end

it "sets the options" do

options = {foo: :bar}

adapter = HTTParty::ConnectionAdapter.new(uri, options)

expect(adapter.options.keys).to include(:verify, :verify\_peer, :foo)

end

end

describe ".call" do

let(:uri) { URI 'http://www.google.com' }

let(:options) { { foo: :bar } }

it "generates an HTTParty::ConnectionAdapter instance with the given uri and options" do

expect(HTTParty::ConnectionAdapter).to receive(:new).with(uri, options).and\_return(double(connection: nil))

HTTParty::ConnectionAdapter.call(uri, options)

end

it "calls #connection on the connection adapter" do

adapter = double('Adapter')

connection = double('Connection')

expect(adapter).to receive(:connection).and\_return(connection)

allow(HTTParty::ConnectionAdapter).to receive\_messages(new: adapter)

expect(HTTParty::ConnectionAdapter.call(uri, options)).to be connection

end

end

describe '#connection' do

let(:uri) { URI 'http://www.google.com' }

let(:options) { Hash.new }

let(:adapter) { HTTParty::ConnectionAdapter.new(uri, options) }

describe "the resulting connection" do

subject { adapter.connection }

it { is\_expected.to be\_an\_instance\_of Net::HTTP }

context "using port 80" do

let(:uri) { URI 'http://foobar.com' }

it { is\_expected.not\_to use\_ssl }

end

context "when dealing with ssl" do

let(:uri) { URI 'https://foobar.com' }

context "uses the system cert\_store, by default" do

let!(:system\_cert\_store) do

system\_cert\_store = double('default\_cert\_store')

expect(system\_cert\_store).to receive(:set\_default\_paths)

expect(OpenSSL::X509::Store).to receive(:new).and\_return(system\_cert\_store)

system\_cert\_store

end

it { is\_expected.to use\_cert\_store(system\_cert\_store) }

end

context "should use the specified cert store, when one is given" do

let(:custom\_cert\_store) { double('custom\_cert\_store') }

let(:options) { {cert\_store: custom\_cert\_store} }

it { is\_expected.to use\_cert\_store(custom\_cert\_store) }

end

context "using port 443 for ssl" do

let(:uri) { URI 'https://api.foo.com/v1:443' }

it { is\_expected.to use\_ssl }

end

context "https scheme with default port" do

it { is\_expected.to use\_ssl }

end

context "https scheme with non-standard port" do

let(:uri) { URI 'https://foobar.com:123456' }

it { is\_expected.to use\_ssl }

end

context "when ssl version is set" do

let(:options) { {ssl\_version: :TLSv1} }

it "sets ssl version" do

expect(subject.ssl\_version).to eq(:TLSv1)

end

end if RUBY\_VERSION > '1.9'

end

context "when dealing with IPv6" do

let(:uri) { URI 'http://[fd00::1]' }

it "strips brackets from the address" do

expect(subject.address).to eq('fd00::1')

end

end

context "specifying ciphers" do

let(:options) { {ciphers: 'RC4-SHA' } }

it "should set the ciphers on the connection" do

expect(subject.ciphers).to eq('RC4-SHA')

end

end if RUBY\_VERSION > '1.9'

context "when timeout is not set" do

it "doesn't set the timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false

)

expect(http).not\_to receive(:open\_timeout=)

expect(http).not\_to receive(:read\_timeout=)

expect(http).not\_to receive(:write\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when setting timeout" do

context "to 5 seconds" do

let(:options) { {timeout: 5} }

describe '#open\_timeout' do

subject { super().open\_timeout }

it { is\_expected.to eq(5) }

end

describe '#read\_timeout' do

subject { super().read\_timeout }

it { is\_expected.to eq(5) }

end

if RUBY\_VERSION >= '2.6.0'

describe '#write\_timeout' do

subject { super().write\_timeout }

it { is\_expected.to eq(5) }

end

end

end

context "and timeout is a string" do

let(:options) { {timeout: "five seconds"} }

it "doesn't set the timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false

)

expect(http).not\_to receive(:open\_timeout=)

expect(http).not\_to receive(:read\_timeout=)

expect(http).not\_to receive(:write\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

end

context "when timeout is not set and read\_timeout is set to 6 seconds" do

let(:options) { {read\_timeout: 6} }

describe '#read\_timeout' do

subject { super().read\_timeout }

it { is\_expected.to eq(6) }

end

it "should not set the open\_timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0

)

expect(http).not\_to receive(:open\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

it "should not set the write\_timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0

)

expect(http).not\_to receive(:write\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when timeout is set and read\_timeout is set to 6 seconds" do

let(:options) { {timeout: 5, read\_timeout: 6} }

describe '#open\_timeout' do

subject { super().open\_timeout }

it { is\_expected.to eq(5) }

end

if RUBY\_VERSION >= '2.6.0'

describe '#write\_timeout' do

subject { super().write\_timeout }

it { is\_expected.to eq(5) }

end

end

describe '#read\_timeout' do

subject { super().read\_timeout }

it { is\_expected.to eq(6) }

end

it "should override the timeout option" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0,

:open\_timeout= => 0,

:write\_timeout= => 0,

)

expect(http).to receive(:open\_timeout=)

expect(http).to receive(:read\_timeout=).twice

if RUBY\_VERSION >= '2.6.0'

expect(http).to receive(:write\_timeout=)

end

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when timeout is not set and open\_timeout is set to 7 seconds" do

let(:options) { {open\_timeout: 7} }

describe '#open\_timeout' do

subject { super().open\_timeout }

it { is\_expected.to eq(7) }

end

it "should not set the read\_timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:open\_timeout= => 0

)

expect(http).not\_to receive(:read\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

it "should not set the write\_timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:open\_timeout= => 0

)

expect(http).not\_to receive(:write\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when timeout is set and open\_timeout is set to 7 seconds" do

let(:options) { {timeout: 5, open\_timeout: 7} }

describe '#open\_timeout' do

subject { super().open\_timeout }

it { is\_expected.to eq(7) }

end

if RUBY\_VERSION >= '2.6.0'

describe '#write\_timeout' do

subject { super().write\_timeout }

it { is\_expected.to eq(5) }

end

end

describe '#read\_timeout' do

subject { super().read\_timeout }

it { is\_expected.to eq(5) }

end

it "should override the timeout option" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0,

:open\_timeout= => 0,

:write\_timeout= => 0,

)

expect(http).to receive(:open\_timeout=).twice

expect(http).to receive(:read\_timeout=)

if RUBY\_VERSION >= '2.6.0'

expect(http).to receive(:write\_timeout=)

end

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

if RUBY\_VERSION >= '2.6.0'

context "when timeout is not set and write\_timeout is set to 8 seconds" do

let(:options) { {write\_timeout: 8} }

describe '#write\_timeout' do

subject { super().write\_timeout }

it { is\_expected.to eq(8) }

end

it "should not set the open timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0,

:open\_timeout= => 0,

:write\_timeout= => 0,

)

expect(http).not\_to receive(:open\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

it "should not set the read timeout" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0,

:open\_timeout= => 0,

:write\_timeout= => 0,

)

expect(http).not\_to receive(:read\_timeout=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when timeout is set and write\_timeout is set to 8 seconds" do

let(:options) { {timeout: 2, write\_timeout: 8} }

describe '#write\_timeout' do

subject { super().write\_timeout }

it { is\_expected.to eq(8) }

end

it "should override the timeout option" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false,

:read\_timeout= => 0,

:open\_timeout= => 0,

:write\_timeout= => 0,

)

expect(http).to receive(:read\_timeout=)

expect(http).to receive(:open\_timeout=)

expect(http).to receive(:write\_timeout=).twice

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

end

context "when max\_retries is not set" do

it "doesn't set the max\_retries" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false

)

expect(http).not\_to receive(:max\_retries=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

context "when setting max\_retries" do

if RUBY\_VERSION >= '2.5.0'

context "to 5 times" do

let(:options) { {max\_retries: 5} }

describe '#max\_retries' do

subject { super().max\_retries }

it { is\_expected.to eq(5) }

end

end

context "to 0 times" do

let(:options) { {max\_retries: 0} }

describe '#max\_retries' do

subject { super().max\_retries }

it { is\_expected.to eq(0) }

end

end

end

context "and max\_retries is a string" do

let(:options) { {max\_retries: "five times"} }

it "doesn't set the max\_retries" do

http = double(

"http",

:null\_object => true,

:use\_ssl= => false,

:use\_ssl? => false

)

expect(http).not\_to receive(:max\_retries=)

allow(Net::HTTP).to receive\_messages(new: http)

adapter.connection

end

end

end

context "when debug\_output" do

let(:http) { Net::HTTP.new(uri) }

before do

allow(Net::HTTP).to receive\_messages(new: http)

end

context "is set to $stderr" do

let(:options) { {debug\_output: $stderr} }

it "has debug output set" do

expect(http).to receive(:set\_debug\_output).with($stderr)

adapter.connection

end

end

context "is not provided" do

it "does not set\_debug\_output" do

expect(http).not\_to receive(:set\_debug\_output)

adapter.connection

end

end

end

context "when uri port is not defined" do

context "falls back to 80 port on http" do

let(:uri) { URI 'http://foobar.com' }

before { allow(uri).to receive(:port).and\_return(nil) }

it { expect(subject.port).to be 80 }

end

context "falls back to 443 port on https" do

let(:uri) { URI 'https://foobar.com' }

before { allow(uri).to receive(:port).and\_return(nil) }

it { expect(subject.port).to be 443 }

end

end

end

end