Язык программирования Ruby, занятие 4

CS 212

19 ноября 2018 г.

План

1. TDD
2. Тестирование с Test Unit
3. Web приложения
4. Коды HTTP
5. Глаголы HTTP
6. REST API

Алгоритм TDD

1. Написать тесты
2. Выполнить тесты и убедиться, что они не проходят
3. Написать минимальный код, который пройдет тесты
4. Выполнить тесты и убедиться, что они проходят
5. Выполнить рефакторинг

Выполнение теста

1. Подготовка UUT (unit under test)
2. Выполнение кода
3. Валидация результатов
4. Очистка

Хорошие практики

1. Короткие итерации
2. Маленькие модули
3. Самодокументируемые тесты
4. Отделяйте общую настройку окружения от частной для теста
5. Отдельные наборы тестов должны тестировать отдельные модули
6. Учитывайте ошибочные негативные срабатывания при тестировании долгих операций
7. Проводите code review
8. Пишите красивый и качественный тестовый код

Плохие практики

1. Тесты, зависящие от состояния, созданного другими тестами
2. Зависимости между тестами
3. Точное тестирование производительности или времени выполнения
4. Проектирование всезнающих "оракулов"
5. Тестирование реализации
6. Медленные тесты

Простейшие тесты

```
1  require "test/unit"
2
3  class TestSequence < Test::Unit::TestCase
4
5  def test_normal
6    assert_equal(1, sequence(0, 1, 3))
7  end
8
9  def test_small_value
10   assert_equal(1, sequence(1, 2, 1))
11 end
12 end
13 end
```

Листинг 1: Простейшие тесты

Asserts

```ruby
assert(false, "This was expected to be true")
assert_equal(expected, actual, text)
assert_block do
  [1, 2, 3].any? { |num| num < 1 }
end
assert_nothing_raised RuntimeError do
  raise Exception #Assertion passes, Exception is
      not a RuntimeError
end

assert_nothing_raised do
  raise Exception #Assertion fails
end
```

Листинг 2: Asserts

Asserts

```
1 assert_respond_to("hello", :reverse) #Succeeds
2 assert_respond_to("hello", :does_not_exist) #Fails
3 assert_send(["Hello world", :include?, "Hello"]) #
     -> pass
4 assert_send(["Hello world", :include?, "Goodbye"])
     # -> fail
```

Листинг 3: Asserts 2

Web приложения

1. Архитектура
2. Uptime
3. Многопользовательское приложение
4. Многопоточное приложение
5. Набор шаблонных действий

Базовый функционал web приложения

1. Диманический рендеринг страниц
2. Хранение данных и доступ к ним
3. Механизмы авторизации
4. Интеграции с другими web приложениями

URL

<схема>://[<логин>[:<пароль>]@]<хост>[:<порт>]]
[/<URL-путь>][?<параметры>][#<якорь>]

Зарезервированные домены:
example, localhost, invlaid, test

Коды HTTP

1. 1XX - информационные
2. 2XX - успех
3. 3XX - перенаправление
4. 4XX - ошибка клиента
5. 5XX - ошибка сервера

Некоторые полезные коды

1. 200 - OK
2. 307,308 - временное и постоянное перенаправление
3. 400 - Ошибка клиента
4. 403 - Запрещено
5. 404 - Не найдено
6. 418 - "Я чайник"
7. 500 - Ошибка сервера
8. 502 - Bad Gateway
9. 503 - Сервер недоступен
10. 504 - Тайм-аут

Глаголы HTTP

1. GET Запрашивает представление ресурса
2. HEAD Запрашивает ресурс так же, как и метод GET, но без тела ответа
3. POST Используется для отправки сущностей к определённому ресурсу
4. PUT Заменяет все текущие представления ресурса данными запроса
5. DELETE Удаляет указанный ресурс.
6. CONNECT Устанавливает "туннель"к серверу, определённому по ресурсу
7. OPTIONS Используется для описания параметров соединения с ресурсом.
8. TRACE Выполняет вызов возвращаемого тестового сообщения с ресурса
9. PATCH Используется для частичного изменения ресурса.

Заголовки запросов

1. General Headers — используются в запросах и ответах.
2. Request Headers — используются только в запросах.
3. Response Headers — используются только в ответах.
4. Entity Headers — сопровождают каждую сущность сообщения.

Как сделать запрос?

Браузер, Postman, IDE RubyMine->Tools->HTTP Client

```
1  GET https://httpbin.org/ip
2  Accept: application/json
3
4  POST https://httpbin.org/post
5  Content-Type: application/json
6
7  {
8  "id": 999,
9  "value": "content"
10 }
```

Листинг 4: Запросы

Как сделать запрос?

```
1 POST https://httpbin.org/post
2 Content-Type: application/x-www-form-urlencoded
3
4 id=999&value=content
5
6 POST https://httpbin.org/post
7 Content-Type: multipart/form-data;
    boundary=WebAppBoundary
```

Листинг 5: Запросы 2

JSON

JSON = Java Script Object Notation Описывает либо ассоциативный массив, либо упорядоченный массив
Типы данных:
Объект — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки « ». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
Массив — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.
Число.
Литералы true, false и null.
Строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки.

Образец JSON

https://jsonlint.com/

```
1  {
2    "firstName": "Ivan",
3    "lastName": "Ivanov",
4    "address": {
5      "streetAddress": "Baker Street, 221b",
6      "city": "London",
7      "postalCode": "101101"
8    },
9    "phoneNumbers": [
10   "812 123-1234",
11   "916 123-4567"
12   ]
13 }
```

Листинг 6: Пример JSON

Существующие подходы к описанию API

1. REST - «Representational State Transfer» - WADL
2. SOAP - Simple Object Access Protocol - WSDL

REST - Требования

1. Клиент-серверная архитектура
2. Отсутствие состояния
3. Возможность кэширования
4. Единообразность интерфейса
5. Многослойность
6. Передача кода по требованию