

# Postman (продолжение)

# Что тестируем

- Открытое API для получения информации о погоде:  
<https://openweathermap.org/current>
- Для работы с приложением нужен ключ APPID
- Используем авторизацию по API Key:

The screenshot shows the Postman interface with the 'Auth' tab selected. The 'Type' is set to 'API Key'. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'. The 'Key' field contains 'APPID', the 'Value' field contains '5a42a44439b4d568f4efd02f51f11cd5', and the 'Add to' dropdown is set to 'Query Params'.

Tab	Status
Params	●
Auth	●
Headers (6)	●
Body	
Pre-req.	
Tests	●
Settings	

**Type**  
API Key

The authorization header will be automatically generated when you send the request.  
[Learn more about authorization](#)

**Key**  
APPID

**Value**  
5a42a44439b4d568f4efd02f51f11cd5

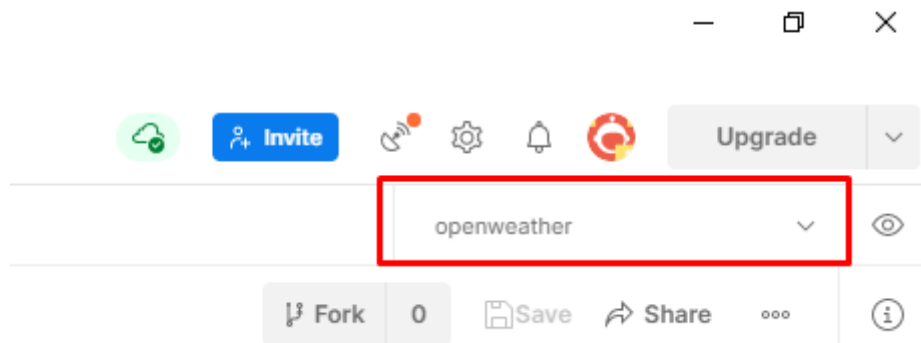
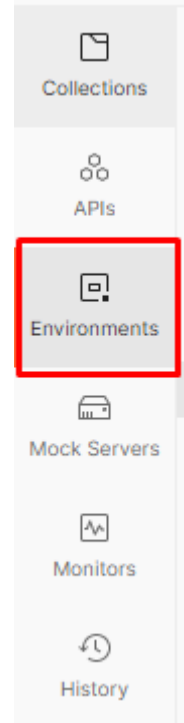
**Add to**  
Query Params

- Протестируем позитивные и негативные кейсы

# Postman Переменные

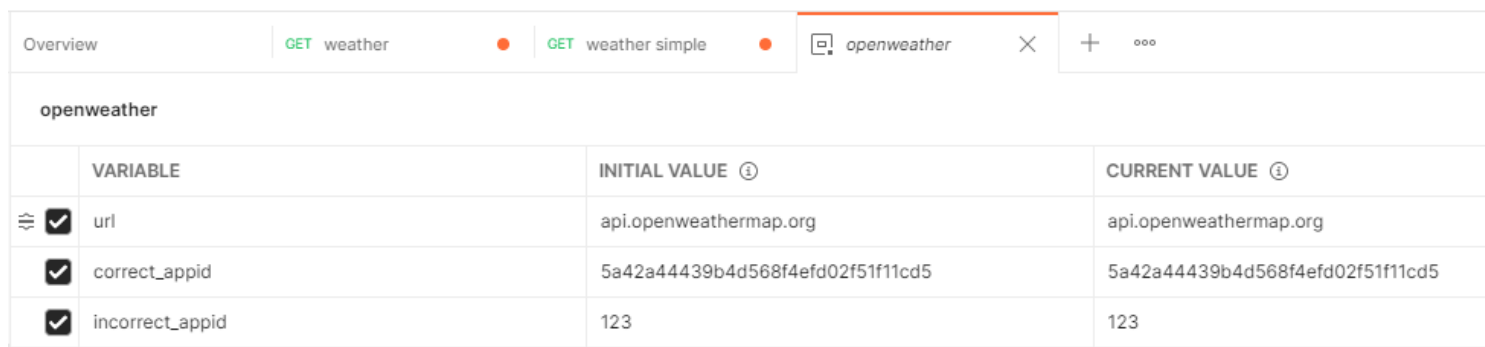
# Переменные в Postman

- Переменные в Postman организуются в окружения (**Environments**)
- Есть глобальные переменные – Globals, доступные во всех окружениях, но чаще используются переменные для конкретного окружения
- **Environments** выбираются в правом верхнем углу окна Postman:



# Переменные в Postman

- Создание переменной:



The screenshot shows the Postman interface with the 'openweather' collection selected. Below the collection name, there is a table of environment variables. The table has four columns: a checkbox, 'VARIABLE', 'INITIAL VALUE', and 'CURRENT VALUE'. Three variables are listed: 'url', 'correct\_appid', and 'incorrect\_appid'. All three have their checkboxes checked and their initial and current values are identical.

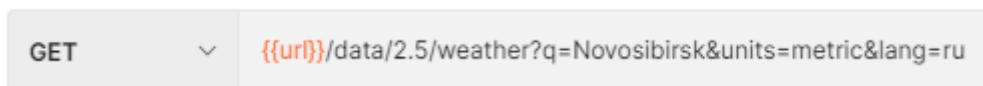
	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	url	api.openweathermap.org	api.openweathermap.org
<input checked="" type="checkbox"/>	correct_appid	5a42a44439b4d568f4efd02f51f11cd5	5a42a44439b4d568f4efd02f51f11cd5
<input checked="" type="checkbox"/>	incorrect_appid	123	123

- Изначально initial value = current value, при выполнении тестов current value может меняться, если мы написали такую логику
- Создание и получение переменных в тесте можно найти в snippets. Например:  
`postman.setEnvironmentVariable("newVar", jsonData.coord.lon);`

# Переменные в Postman

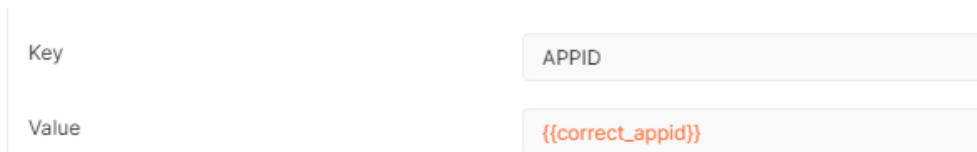
- Для использования переменной не забудьте сначала выбрать окружение, в котором сохранена переменная (!)
- Где можно использовать переменные:

- В URL:



GET ▼ `{{url}}/data/2.5/weather?q=Novosibirsk&units=metric&lang=ru`

- В заголовках
- Во вкладке Authorization



Key	APPID
Value	<code>{{correct_appid}}</code>

- В теле запроса
- В тестах в формате `pm.environment.get("variable_key")`
- <https://www.software-testing.ru/library/testing/testing-automation/2992-using-variables-in-postman>

# Практика 1

- Создадим переменные `url`, `correct_appid` и `incorrect_appid`
- Реализуем позитивные и негативные проверки на авторизацию в приложении
- Протестируем различные параметры: `q`, `metrics`, `lang` и др.
- `{{url}}/data/2.5/weather?q=Novosibirsk&units=metric&lang=ru`

## Практика 2

- Создадим переменные city, lat и lon
- Выполним запрос погоды для city и сохраним значения полученных в ответе lat и lon
- Запросим погоду для сохраненных lat и lon
- Проверим, что получаемый город = city
- `{{url}}/data/2.5/weather?lat={{lat}}&lon={{lon}}`



# Postman

## Написание тестов на JS

# Тесты на JS в Postman

- Для написания тестов в Postman используется библиотека **Chai.js** <https://www.chaijs.com/>. Также можно использовать Node.js
- Тесты могут быть написаны для отдельных запросов, папок или целых коллекций
- Тесты исполняются после получения ответа
- Для написания тестов существуют удобные snippets
- Документация Postman  
<https://learning.postman.com/docs/writing-scripts/script-references/test-examples/>
- Генерация схемы JSON  
<https://www.jsonschema.net/home>

# Тесты на статус код

- Существует очень много способов проверить статус код ответа с использованием библиотеки Chai.js
- Разберем некоторые из них
- Тест представляет из себя функцию, которая имеет имя и возвращает boolean (true/false)

```
pm.test("Status code is 200", function () {  
  pm.response.to.have.status(200);  
});
```

- pm.response – это объект ответа в Chai.js
- Тест на статус код в сниппетах: Status code: Code is 200
- <https://learning.postman.com/docs/writing-scripts/script-references/test-examples/#testing-status-codes>

# Тесты на статус код

- Существует также объект `pm.expect`, который позволяет писать более разнообразные тесты
- Проверка, что статус код является одним, из набора:

```
pm.test("Successful GET request", function () {  
  pm.expect(pm.response.code).to.be.oneOf([200, 201, 202,  
  203]);  
});
```

- В snippets: `Status code: Successful POST request`
- Snippet для проверки **текста в статус коде**:  
`Status code: Status code name has string`  
`pm.response.to.have.status("Unauthorized");`

# Тесты на статус код

- Другие варианты проверки статус кода:
- `pm.expect(pm.response.code).to.eq(200);` / `to.equal(200);`
- `pm.expect(pm.response.code).to.be.below(300);`
- `pm.expect(pm.response.code).to.be.above(100);`
- `pm.response.to.be.info;` (для 1xx кодов)
- `pm.response.to.be.success;` (для 2xx кодов)
- `pm.response.to.be.ok;` (для 200 OK)
- `pm.response.to.be.redirection;` (для 3xx кодов)
- `pm.response.to.be.clientError;` (для 4xx кодов)
- `pm.response.to.be.serverError;` (для 5xx кодов)
- `pm.response.to.be.error;` (для любых ошибок)

# Тесты на статус код

- Другие варианты проверки статус кода:
- <https://www.chaijs.com/api/bdd/>
- `pm.response.to.not.have.status(300);`
- `pm.expect(pm.response.code).to.be.a('number');`
- `pm.expect(pm.response.code === 200).to.be.true;`
- `pm.expect(pm.response.code === 200).to.not.be.null;`
- `pm.expect(pm.response.code).to.be.at.least(200);`
- `pm.expect(pm.response.code).to.be.at.most(300);`
- `pm.expect(pm.response.code).to.be.within(100, 300);`

# Тесты на статус код

- Проверки статус кода с помощью библиотеки **Node.js**
- <https://nodejs.org/api/assert.html>
- `const assert = require('assert').strict;` // импорт библиотеки  
`assert.equal(pm.response.code, 200);`
- `const assert = require('assert').strict;` // импорт библиотеки  
`assert(pm.response.code === 200);`

# Проверка response body

## Что изучим:

- Тесты из snippets
- Переменные var, let, const
- Достаем значение из простого JSON
- Достаем значение из сложного дерева JSON
- Условный оператор в тестах
- Валидируем схему JSON



# Проверка response body. Snippets

- Response body: JSON value check

```
pm.test("Your test name", function () {  
    var jsonData = pm.response.json();  
    pm.expect(jsonData.value).to.eql(200);  
});
```

Объявление  
переменной и  
присваивание  
ей значения:  
результат  
выполнения  
функции json()

Можно вывести значение pm.response в консоль:  
console.log(pm.response);

```
▼ {id: "9df148ff-ea68-47d0-bf26-b58b8b1071ae", status: "OK", code: 200...}  
  id: "9df148ff-ea68-47d0-bf26-b58b8b1071ae"  
  status: "OK"  
  code: 200  
  ► header: [9]  
  ► stream: {...}  
  cookie: [0]  
  responseTime: 139  
  responseSize: 475
```

Можно  
вынести эту  
переменную  
на уровень  
выше, она  
нужна для  
многих тестов

# Проверка response body. Snippets

Также можно вывести значение `pm.response.json()` в консоль:  
`console.log(pm.response.json());`

- Это объект JSON
- Порядок полей в объект не строгий, может отличаться
- Обращение – по ключу, не привязывайтесь в тестах к порядку полей (!)
- `console.log` должен быть написан ДО `pm.exhrest`, так как `pm.exhrest` - это окончание теста, функция завершится

```
▼ {coord: {...}, weather: [1], base: "stations"...}
  ▼ coord: {...}
    lon: 82.9344
    lat: 55.0411
  ▼ weather: [1]
    ▼ 0: {...}
      id: 800
      main: "Clear"
      description: "ясно"
      icon: "01n"
    base: "stations"
  ▼ main: {...}
    temp: 18.61
    feels_like: 18.3
    temp_min: 18.61
    temp_max: 18.61
    pressure: 1003
    humidity: 68
    visibility: 10000
  ▼ wind: {...}
    speed: 2
    deg: 360
  ► clouds: {...}
    dt: 1625933077
  ► sys: {...}
    timezone: 25200
    id: 1496747
    name: "Новосибирск"
    cod: 200
```

# Переменные в JS

`var x = 3;`

ключевое  
слово

имя  
переменной

значение переменной,  
может быть результатом  
вычисления

- Имена в формате camelCase
- “=” – это присваивание, “==” – нестрогое сравнение (с приведением типа: “2”=2), “===” – строгое сравнение (“2”!=2)
- Ключевые слова для переменных в JS:
  - **var** – переменная «видна» до конца функции
  - **let** – переменная «видна» внутри одного блока {...}
  - **const** – константа, нельзя изменить значение
  - **(ничего)** – переменная «видна» везде, глобальная

# Проверка response body. Snippets

- Проверим id, name и timezone из тела ответа:

```
pm.expect(jsonData.id).to.eql(1496747);  
pm.expect(jsonData.name).to.eql("Novosibirsk");  
pm.expect(jsonData.timezone).to.eql(25200);
```

# Проверка response body. Snippets

- Еще один сниппет для проверки тела ответа:  
Response body: Contains string
- Проверим текст сообщения для невалидного APPID:  

```
pm.test("Error response is returned", function () {  
  pm.expect(pm.response.text()).to.include("Invalid API key");  
});
```

# Работа с простым JSON

- Тест для невалидного APPID возвращает простой JSON:

```
{  
  "cod": 401,  
  "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."  
}
```

- Обращаемся просто по ключу поля:

```
pm.test("Check error message", function () {  
  var jsonData = pm.response.json();  
  pm.expect(jsonData.message).to.eql("Invalid API key.  
  Please see http://openweathermap.org/faq#error401  
  for more info.");  
});
```

- Можно обращаться так: **jsonData["message"]**
- Так нельзя: **jsonData[1] (!)**

# Работа с массивом в JSON

- Поле `weather` представляет собой массив объектов в `[]`
- Для обращения к полям объектов в массиве нужно указать индекс элемента массива, а потом поле:

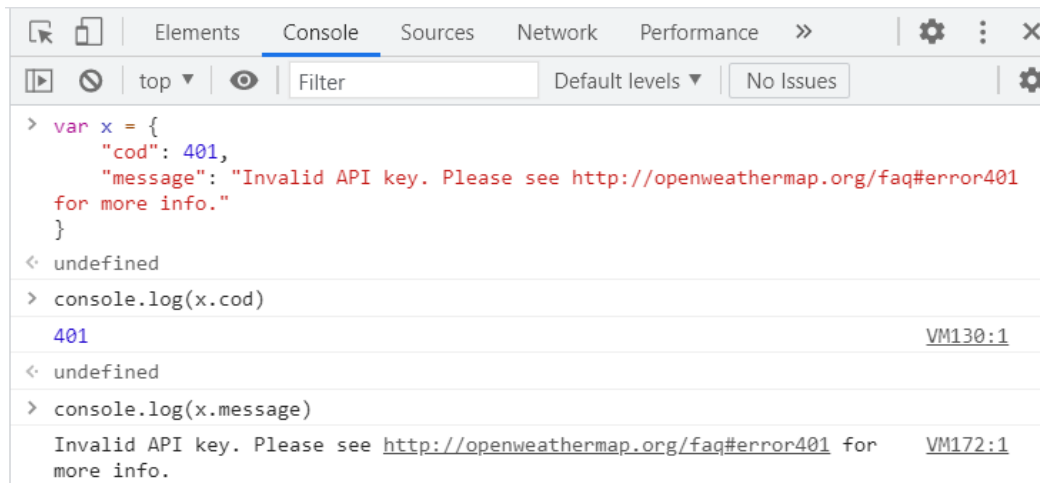
```
pm.test("Check weather description",  
function () {  
  var jsonData = pm.response.json();  
  pm.expect(jsonData.weather[0].main)  
    .to.eql("Clouds");  
});
```

```
{  
  "coord": {  
    "lon": 82.9344,  
    "lat": 55.0411  
  },  
  "weather": [  
    {  
      "id": 800,  
      "main": "Clear",  
      "description": "ясно",  
      "icon": "01n"  
    }  
  ],  
  "base": "stations",  
  "main": {  
    "temp": 16.61,  
    "feels_like": 16.34,  
    "temp_min": 16.61,  
    "temp_max": 16.61,  
    "pressure": 1003,  
    "humidity": 77  
  },  
  ...  
}
```

- Индексы в массиве начинаются с 0

# Работа в консоли

- Чтобы проверить себя при работе со сложными объектами и массивами, можно использовать консоль
- Инструменты разработчика F12 → Console
- Сначала создаем объект или массив, а затем выводим в консоль данные из него



```
> var x = {  
  "cod": 401,  
  "message": "Invalid API key. Please see http://openweathermap.org/faq#error401  
  for more info."  
}  
< undefined  
> console.log(x.cod)  
401  
< undefined  
> console.log(x.message)  
Invalid API key. Please see http://openweathermap.org/faq#error401 for  
more info.
```

- Вывести все поля объекта: `Object.keys(x);`



# Проверка нескольких полей сразу

- Тест для невалидного APPID возвращает простой JSON:

```
"cod": 401,  
"message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
```

- Проверяем все поля сразу:  
`pm.test("All fields are correct", function () {  
 var jsonData = pm.response.json();  
 pm.expect(jsonData).to.eql({ "cod": 401,  
 "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info." }); });`
- Можно использовать **to.include** для неточного равенства.
- Для проверок вложенных объектов – **to.deep.include**

# Проверка нескольких полей сразу

- Тест для невалидного APPID возвращает простой JSON:

```
"cod": 401,  
"message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
```

- Проверяем наличие всех полей сразу другим способом:  
`pm.test("ALL fields are present", function () {  
 var jsonData = pm.response.json();  
 pm.expect(jsonData).to.have.all.keys("cod", "message");  
});`
- Можно использовать `any` для неточного равенства:  
`pm.expect(jsonData).to.have.any.keys ("cod");`
- Аналогичная функция: **members**

# Проверка через property

- Тест для невалидного APPID возвращает простой JSON:

```
"cod": 401,  
"message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
```

- Проверяем наличие поля message:  
`pm.test("Presence of the message field", function () {  
 var jsonData = pm.response.json();  
 pm.expect(jsonData).to.have.property("message");  
});`
- Проверяем наличие поля + значение:  
`pm.expect(jsonData).to.have.property("cod", 401);`
- Для проверок вложенных объектов – **to.have.deep. Property**
- [https://www.chaijs.com/api/bdd/#method\\_deep](https://www.chaijs.com/api/bdd/#method_deep)

# Проверка типа

- Поле weather представляет собой массив объектов в []
- Проверим это:

```
pm.test("Check weather type is array",  
function () {  
  var jsonData = pm.response.json();  
  pm.expect(jsonData.weather)  
    .to.be.an.instanceOf(Array);  
});
```

- Проверим, что массив не пустой:  
pm.expect(jsonData.weather).to.be.an.  
instanceOf(Array).that.is.not.empty;

```
{  
  "coord": {  
    "lon": 82.9344,  
    "lat": 55.0411  
  },  
  "weather": [  
    {  
      "id": 800,  
      "main": "Clear",  
      "description": "ясно",  
      "icon": "01n"  
    }  
  ],  
  "base": "stations",  
  "main": {  
    "temp": 16.61,  
    "feels_like": 16.34,  
    "temp_min": 16.61,  
    "temp_max": 16.61,  
    "pressure": 1003,  
    "humidity": 77  
  },  
  ...  
}
```

# Условный оператор if

- `if (условие) {  
    // код  
}`

Код внутри { }  
выполняется, если  
условие истинно

- В качестве условия может быть логическое выражение (==, >, < и т.д.)
- Если передать строку, не-нулевое число или true в качестве условия, то вернется **true**
- Если передать пустую строку, false, 0, NaN, undefined или null – вернется **false**
- Простой пример в консоли браузера/Postman'а:

```
> if(3>1){console.log("3 more than 1");}  
3 more than 1
```

# Условный оператор if

- **if** (условие) {

// код

} **else** {

// другой код

}

Код выполняется, если  
условие истинно

Код выполняется, если  
условие ложно

- В JavaScript также есть конструкция:

**if** (условие) {

// код

} **else if** (условие) {

// другой код

} **else** {

// третий код

}

- Может быть сколько угодно **else if** веток

# Условный оператор if

- В качестве условия может быть удобно использовать проверку длины массива:
- `pm.expect(jsonData.weather.length == 1).to.be.true;`
- Также эту проверку можно использовать в качестве отдельного теста

# Условный оператор практика

- Тест для невалидного APPID возвращает простой JSON:

```
"cod": 401,  
"message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
```

- Можно написать тест с проверкой условия  
`if (cod == 401) {`  
    // message должен быть равен "Invalid API key.  
    Please see <http://openweathermap.org/faq#error401> for more info."  
`}`  
  
`if (jsonData.cod == 401) {`  
    pm.expect(jsonData.message).toEqual("Invalid API key. ... ");  
`}`



# Условный оператор практика

- Поля 'id' и 'main' в объекте weather связаны с друг с другом
- Для каждой группы айдишников значение поля 'main' будет одинаковым
- <https://openweathermap.org/weather-conditions>
- Реализуем проверки поля 'main' в зависимости от поля 'id'  

```
if (weatherId==800) {  
    pm.expect(jsonData.weather[0].main).to.eql("Clear"); }  
  
if (weatherId>=801 && weatherId<=804) {  
    pm.expect(jsonData.weather[0].main).to.eql("Clouds"); }
```

```
"weather": [  
  {  
    "id": 803,  
    "main": "Clouds",  
    "description": "broken clouds",  
    "icon": "04d"  
  }  
],
```

# Условный оператор if

- Когда может быть полезен условный оператор if:
  - Когда есть зависимость между полями
  - Когда тестов много и нужно запускать не все из них, а только те, которые удовлетворяют некоторому условию
  - Чтобы избегать false failed тестов: проверять тело ответа, только если запрос прошел успешно

# Генерация случайных данных

- Динамические переменные в Postman:  
<https://learning.postman.com/docs/writing-scripts/script-references/variables-list/>
- Можно использовать в Pre-Request Scripts и в Tests
- Можно использовать в параметрах запроса.  
Для этого нужно сгенерировать значения в Pre-Request Scripts, сохранить их в переменные и использовать как обычные переменные в query string'e

# Валидация схемы JSON

- **JSON схема** – это описание некоторого JSON, точный набор полей с типами данных
- Обычно валидация схемы JSON проводится в рамках smoke-тестов
- Книга по JSON схемам: <https://json-schema.org/understanding-json-schema/index.html>
- Генерация схемы JSON <https://www.jsonschema.net/home>

```
"$schema": "http://json-schema.org/draft-07/schema",  
"$id": "http://example.com/example.json",  
"type": "object",  
"title": "The root schema",  
"description": "The root schema comprises the entire JSON document.",  
"default": {},  
"examples": [↔],  
"required": [↔],  
"properties": {↔},  
"additionalProperties": true
```

# Валидация схемы JSON

- **Обязательные поля схемы:**

```
const schema = {  
  "required": [  
    "cod",  
    "message"  
  ],  
  "properties": {  
    "cod": {  
      "type": "integer"  
    },  
    "message": {  
      "type": "string"  
    }  
  }  
};
```

## Примеры типов:

- object
- array
- integer
- number (целые или вещ. числа)
- string
- boolean
- null

## Валидация емэйла:

```
"type": "string",  
"format" : "email"
```

# Валидация схемы JSON

- **Вспомогательные поля схемы** (не используются при валидации схемы):

```
"$schema": "http://json-schema.org/draft-07/schema",  
"$id": "http://example.com/example.json",  
"title": "The root schema",  
"description": "The root schema comprises the entire JSON",  
"default": {},           // дефолтные значения  
"examples": []           // пример валидной схемы  
  
"$comment": "comment"    // комментарии
```

# Валидация схемы JSON

- Как использовать в Postman:  
<https://learning.postman.com/docs/writing-scripts/script-references/test-examples/#validating-response-structure>

- 2 варианта: с помощью валидаторов tv4 или ajv

```
const schema = {
  "items": {
    "type": "boolean"
  }
};

const data1 = [true, false];
const data2 = [true, 123];

pm.test('Schema is valid', function() {
  pm.expect(tv4.validate(data1, schema)).to.be.true;
  pm.expect(tv4.validate(data2, schema)).to.be.true;
});
```

```
const schema = {
  "properties": {
    "alpha": {
      "type": "boolean"
    }
  }
};

pm.test('Schema is valid', function() {
  pm.response.to.have.jsonSchema(schema);
});
```

- Схему можно писать самим, а можно использовать автогенераторы
- Удобный вывод ошибок при валидации схем в консоль:  
`console.log(tv4.error.dataPath);`

# **Postman Runners и Monitors**



# Runner в Postman

- Запуск всех тесты в коллекции:
- Можно выставить количество повторений, задержку запуска, а также подгрузить данные из внешнего файла:

Iterations

Delay  ms

Data

REST Openweathermap



Share collection

Run collection

Edit

Add request

Add folder

Monitor collection

Mock collection

Create a fork

Create Pull Request

Merge changes

View documentation

Rename Ctrl+E

Duplicate Ctrl+D

Export

Manage roles

# Runner в Postman

- Результат прогона автотестов:

REST Openweathermap openweather, just now

## RUN SUMMARY

			1
GET	weather simple	0   0	
GET	weather simple units	0   0	
GET	weather var positive	0   0	
GET	weather var negative	0   0	
▶ GET	weather	16   3	×
▶ GET	weather Invalid APPID	1   1	×
▶ GET	weather NEW	7   0	
▶ GET	weather NEW Invalid APPID	11   1	×

- Описание ошибок:

All Tests Passed (35) Failed (5)

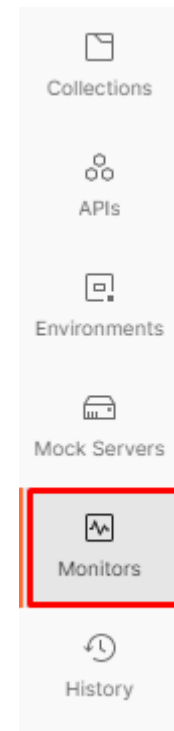
## Iteration 1

GET weather {{url}}/data/2.5/weather?q=Novosibirsk&units=metric&lang=ru / weather

- Fail Chaijs assert Status code is 200 | ReferenceError: assert is not defined
- Fail Your test name | AssertionError: expected undefined to deeply equal 100
- Fail Check id value in response body | AssertionError: expected 'Новосибирск' to deeply equal 'Novosibirsk'

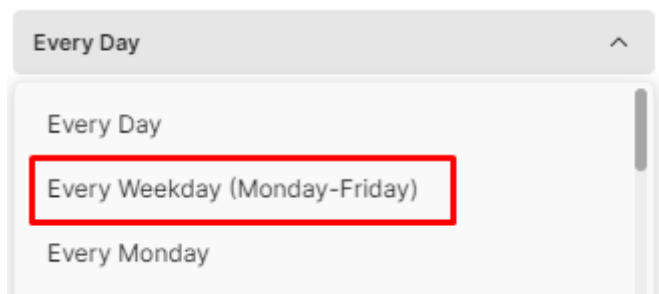
# Мониторы в Postman

- С помощью мониторов можно настроить регулярный запуск автотестов
- Результаты прогонов можно присылать на заданный емэйл
- Запуск происходит удаленно, независимо от того, залогинены ли вы на момент запуска автотестов или нет



# Мониторы в Postman

- Выбираем коллекцию, которую будем запускать
- Указываем environment, переменные которого будут использоваться при запуске
- Настраиваем регулярный запуск. Например, можно выставить ежедневный запуск по рабочим дням:



## Create a monitor

Monitor name

Collection

Select a collection



Collection version tag

Select a version tag for this collection



Environment

No Environment



Run this monitor

Week Timer



Every Day



1:00 PM



# Мониторы в Postman

- Указываем емэйл, на который будут присланы нотификации с результатами прогонов тестов (до 5 емэйлов)
- Можно повторять прогон при падении и выставить задержки между вызовами

## Regions

☒ Automatically select region

☐ Manually select region

☒ Receive email notifications for run failures and errors

e.p.balakina@mail.ru

Add another recipient email

You can add up to 5 notification recipients

Stop notifications after  consecutive failures

☐ Retry if run fails (This might affect your billing.)

☐ Set request timeout

☐ Set delay between requests

☐ Follow redirects

☐ Disable SSL validation

# Дом. задание «Postman. Openweather»

- Открытое API для получения информации о погоде:  
<https://openweathermap.org/current>
- Зарегистрируйтесь на сайте и получите ключ APPID
- Создайте коллекцию с запросами (**не менее 5 запросов**)
- Реализуйте позитивные и негативные проверки
- Протестируйте различные параметры (q, id, zip, units, lang)
- Создайте окружение openweather и переменные для этого окружения (url, correct\_appid, incorrect\_appid). Используйте переменные в запросах
- Напишите тесты на JS: проверка кодов ответов, проверка error message (**не менее 5 проверок**)

# Дом. задание «Postman. Openweather»

- **Задание на оператор IF:**
- Написать тесты, которые проверяют, что `weather.main` соответствует `weather.id`
- Например, для `weather.id = 800`: `weather.main = "Clear"`
- Описание кодов:
- <https://openweathermap.org/weather-conditions>

# Дом. задание «Postman. Openweather»

- **Задание на динамические переменные:**
- Сгенерировать значения широты и долготы
- Использовать их в запросе
- Написать тест, который проверяет вернул ли запрос данные для существующего города (поле name)
- **Задание на проверку схемы json:**
- Напишите тесты на проверку схемы json-ответа для позитивного кейсы получения погоды по заданному городу