

# JSON+Go in practice

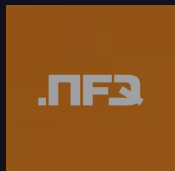
Aurelijus Banelis



# Aurelijus Banelis

Software developer  
[aurelijus.banelis.lt](mailto:aurelijus.banelis.lt)  
[aurelijus@banelis.lt](mailto:aurelijus@banelis.lt)

PGP public key      rsa2048/ **539B6203**  
Key fingerprint = 130D C446 1F1A 2E50 D6E3  
                    3DA8 3202 05E7 539B 6203



# Using JSON with Go in practice

**JSON is a *simple*  
data transfer format**

[illegible]

# The JavaScript Object Notation (JSON) Data Interchange Format

**[http://seriot.ch/parsing\\_json.php](http://seriot.ch/parsing_json.php)**

[illegible]

# Communication

**Naming**

**Versioning**

**Validating**

# Development

**Acceptance tests**

**Unit tests**

**Logging**



# Communication

**Naming**

**Versioning**

**Validating**

# Development

**Acceptance tests**

**Unit tests**

**Logging**



```
{  
  "NetworkID": "b5a465c4ddf1",  
  "EndpointID": "5483c7bbf04",  
  "Gateway": "172.20.0.1",  
  "IPAddress": "172.20.0.4",  
  "IPPrefixLen": 16  
}
```

```
{  
  "networkId": "b5a465c4ddf1",  
  "endpointId": "5483c7bbf04",  
  "gateway": "172.20.0.1",  
  "ipAddress": "172.20.0.4",  
  "ipPrefixLen": 16  
}
```

Problem: I like to name it this way

JSON is all about communication  
Which part is written by Go and  
not-Go programmer

```
{  
  "NetworkID": "b5a465c4ddf1",  
  "EndpointID": "5483c7bbf04",  
  "Gateway": "172.20.0.1",  
  "IPAddress": "172.20.0.4",  
  "IPPrefixLen": 16  
}
```

# go lint

struct field

NetworkId should  
be NetworkID

```
{  
  "networkId": "b5a465c4ddf1",  
  "endpointId": "5483c7bbf04",  
  "gateway": "172.20.0.1",  
  "ipAddress": "172.20.0.4",  
  "ipPrefixLen": 16  
}
```

```
{  
  "NetworkID": "b5a465c4ddf1",  
  "EndpointID": "5483c7bbf04",  
  "Gateway": "172.20.0.1",  
  "IPAddress": "172.20.0.4",  
  "IPPrefixLen": 16  
}
```

# go lint

struct field

NetworkId should  
be NetworkID

```
{  
  "networkId": "b5a465c4ddf1",  
  "endpointId": "5483c7bbf04",  
  "gateway": "172.20.0.1",  
  "ipAddress": "172.20.0.4",  
  "ipPrefixLen": 16  
}
```

# Google JSON Style Guide

Property names  
must be  
camel-cased

```
{
  "NetworkID": "b5a465c4ddf1",
  "EndpointID": "5483c7bbf04",
  "Gateway": "172.20.0.1",
  "IPAddress": "172.20.0.4",
  "IPPrefixLen": 16
}
```

```
{
  "networkId": "b5a465c4ddf1",
  "endpointId": "5483c7bbf04",
  "gateway": "172.20.0.1",
  "ipAddress": "172.20.0.4",
  "ipPrefixLen": 16
}
```

```
import "encoding/json"
```

```
type structureGo struct {
  NetworkID    string
  EndpointID   string
  Gateway      string
  IPAddress    string
  IPPrefixLen  int
}
```

```
import "encoding/json"
```

```
type structureJs struct {
  NetworkID    string `json:"networkId"`
  EndpointID   string `json:"endpointId"`
  Gateway      string `json:"gateway"`
  IPAddress    string `json:"ipAddress"`
  IPPrefixLen  int    `json:"ipPrefixLen"`
}
```

<https://play.golang.org/p/UEMKMI6tC2>

<https://play.golang.org/p/a9DZ2eze7Y>

Need to add struct tags for each property  
Tags for all structures?  
We are lazy...

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress     string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error)  
{  
    value := reflect.ValueOf(s).Elem()  
    for i := 0; i < value.NumField(); i++ {  
        ...  
    }
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!
```

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Manual

# Dynamic

# Generated

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress    string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"  
  
func (s *st) MarshalJSON() ([]byte, error) {  
    {  
        value := reflect.ValueOf(s).Elem()  
        for i := 0; i < value.NumField(); i++ {  
            ...  
        }  
    }  
}
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!  
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Manual

Well known Boring

# Dynamic

# Generated

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkID    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway       string `json:"gateway"`  
    IPAddress     string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error)  
{  
    value := reflect.ValueOf(s).Elem()  
    for i := 0; i < value.NumField(); i++ {  
        ...  
    }  
}
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!
```

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Manual

Well known Boring

# Dynamic

Runtime errors Automated

# Generated



```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress    string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error) {  
    {  
        value := reflect.ValueOf(s).Elem()  
        for i := 0; i < value.NumField(); i++ {  
            ...  
        }  
    }  
}
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!
```

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Manual

Well known Boring

# Dynamic

Runtime errors Automated

# Generated

Compile time check Learning curve

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress    string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error)  
{  
    value := reflect.ValueOf(s).Elem()  
    for i := 0; i < value.NumField(); i++ {  
        ...  
    }  
}
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!
```

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

```
[  
  {  
    "type": "image",  
    "attributes": {  
      "path": "http://tny.im/ajb"  
    }  
  },  
  {  
    "type": "meetup",  
    "attributes": {  
      "language": "Go",  
      "date": "2017-09-27"  
    }  
  }  
]
```

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress    string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error)  
{  
    value := reflect.ValueOf(s).Elem()  
    for i := 0; i < value.NumField(); i++ {  
        ...  
    }  
}
```

*// THIS IS AUTO GENERATED - DO NOT EDIT!*

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Polymorphism

```
[  
  {  
    "type": "image",  
    "attributes": {  
      "path": "http://tny.im/ajb"  
    }  
  },  
  {  
    "type": "meetup",  
    "attributes": {  
      "language": "Go",  
      "date": "2017-09-27"  
    }  
  }  
]
```

```
type polymorphic struct {  
    Type      string  
    Attributes iAttribute  
}  
type iAttribute interface {  
    Type() string  
}
```

Too much magic for generated code.  
Go struct tags – lower learning curve

```
import "encoding/json"
```

```
type structureJs struct {  
    NetworkId    string `json:"networkId"`  
    EndpointID   string `json:"endpointId"`  
    Gateway      string `json:"gateway"`  
    IPAddress    string `json:"ipAddress"`  
    IPPrefixLen  int     `json:"ipPrefixLen"`  
}
```

```
import "encoding/json"  
import "reflect"
```

```
func (s *st) MarshalJSON() ([]byte, error) {  
    {  
        value := reflect.ValueOf(s).Elem()  
        for i := 0; i < value.NumField(); i++ {  
            ...  
        }  
    }  
}
```

```
// THIS IS AUTO GENERATED - DO NOT EDIT!
```

```
type SomeGeneratedDTO struct {  
    ID          CommonGenerated.ID  
    `json:"id"`  
    ParentID    CommonGenerated.ID  
    `json:"parentId"`  
    Title       CommonGenerated.Translation  
    `json:"title"`  
}
```

# Manual

Well known Boring

# Dynamic

Runtime errors Automated

# Generated

Compile time check Learning curve

# Communication

**Naming**

**Versioning**

**Validating**

# Development

**Acceptance tests**

**Unit tests**

**Logging**

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Logging

# Deployment is not instant

Problem about API versioning is

1746  
1810  
1815

There will always be old clients expecting old API  
Time to retry or to reload configuration

2017-09-19 15:00



```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

# Optional

# Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

## Optional

## Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

```
type speaker0_0_1 struct {  
    Name string `json:"name"`  
}
```

```
type speaker0_2_0 struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    HomePage  string `json:"homePage"`  
}
```

```
type speaker0_3_0 struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    LinkedIn  string `json:"linkedIn"`  
}
```

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {
    ToDomainObject() (speaker, error)
}
```

```
type speakerData struct {
    Name      string `json:"name"`
    Topic     *string `json:"topic,omitempty"`
    HomePage  *string
    `json:"homePage,omitempty"`
    LinkedIn  string `json:"linkedIn,omitempty"`
}
```

```
compatibility := struct {
    HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatibility)
json.Unmarshal(payload, &fields)
```

```
type speaker0_0_1 struct {
    Name string `json:"name"`
}
```

```
type speaker0_2_0 struct {
    Name      string `json:"name"`
    Topic     string `json:"topic"`
    HomePage  string `json:"homePage"`
}
```

```
type speaker0_3_0 struct {
    Name      string `json:"name"`
    Topic     string `json:"topic"`
    LinkedIn  string `json:"linkedIn"`
}
```

```
type speaker struct {
    Name      string
    Topic     string
    LinkedIn  string
}
```

```
func (s *speaker0_3_0) ToDomainObject() (speaker, error) {
    return speaker{
        Name:    s.Name,
        Topic:   s.Topic,
        LinkedIn: s.Linkedin,
    }, nil
}
```

Clean and functional  
But requires a lot of repeated code

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

## Optional

## Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

# Optional

# Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {
    ToDomainObject() (speaker, error)
}
```

```
type speakerData struct {
    Name      string `json:"name"`
    Topic     *string `json:"topic,omitempty"`
    HomePage *string
    `json:"homePage,omitempty"`
    LinkedIn  string `json:"linkedIn,omitempty"`
}
```

```
compatibility := struct {
    HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatibility)
json.Unmarshal(payload, &fields)
```

Topic string **`json:"topic"`**  
Topic \*string **`json:"topic"`**

```
{"topic":""}
{}
```

Hard to differentiate between default and not provided field  
Hack like solution – provide custom default value as a marker



```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
    json.Unmarshal(payload, &latest)  
    json.Unmarshal(payload, &compatibility)  
    json.Unmarshal(payload, &fields)
```

Topic string **`json:"topic"`**  
Topic \*string **`json:"topic"`**

```
{"topic":""}  
{}
```

Hard to differentiate between default and not provided field  
Hack like solution – provide custom default value as a marker

```
markerForNotProvided := "\x00"  
data := speakerData{  
    Name: markerForNotProvided,  
}  
req.DecodeJsonPayload(&data)
```

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {
    ToDomainObject() (speaker, error)
}
```

```
type speakerData struct {
    Name      string `json:"name"`
    Topic     *string `json:"topic,omitempty"`
    HomePage *string
    `json:"homePage,omitempty"`
    LinkedIn  string `json:"linkedIn,omitempty"`
}
```

```
compatibility := struct {
    HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatibility)
json.Unmarshal(payload, &fields)
```

Topic string `json:"topic"`  
Topic \*string `json:"topic"`

Attributes **map**[string]string

```
{"topic":""}
```

```
{
  {"Attributes":null}
  {"Attributes":{}}
  {"Attributes":{"field":"value"}}
```

Default values of  
pointers (structs,  
maps, interfaces) can  
lead to unexpected  
errors

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {
    ToDomainObject() (speaker, error)
}
```

```
type speakerData struct {
    Name      string `json:"name"`
    Topic     *string `json:"topic,omitempty"`
    HomePage *string
    `json:"homePage,omitempty"`
    LinkedIn  string `json:"linkedIn,omitempty"`
}
```

```
compatibility := struct {
    HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatibility)
json.Unmarshal(payload, &fields)
```

Topic string `json:"topic"`  
Topic \*string `json:"topic"`

Attributes **map**[string]string

```
{"topic":""}
```

```
{
  "Attributes":null
  "Attributes":{}
  "Attributes":{"field":"value"}
```

Default values of  
pointers (structs,  
maps, interfaces) can  
lead to unexpected  
errors

```
payload := &struct{ Attributes map[string]string }{}
json.Unmarshal([]byte("{}"), &payload)
payload.Attributes["new"] = "value"
```

**panic: assignment to entry in nil map**

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

# Optional

Fast to change Null values Pointer errors

# Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
    json.Unmarshal(payload, &latest)  
    json.Unmarshal(payload, &compatibility)  
    json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

# Optional

Fast to change Null values Pointer errors

# Compatible

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

```
type speakerLatest struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    LinkedIn  string `json:"linkedIn"`  
}
```

```
map[string]json.RawMessage{}
```

```
compatibility := struct {  
    HomePage string  
    `json:"homePage"`  
}
```

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

```
type speakerLatest struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    LinkedIn  string `json:"linkedIn"`  
}
```

```
map[string]json.RawMessage{}
```

```
compatibility := struct {  
    HomePage string  
    `json:"homePage"`  
}
```

```
if _, exists := fields["linkedIn"]; !exists &&  
strings.HasPrefix(compatibility.HomePage,  
linkedInPrefix) {  
    latest.LinkedIn =  
(compatibility.HomePage)[len(linkedInPrefix):]  
}
```



```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

```
type speakerLatest struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    LinkedIn  string `json:"linkedIn"`  
}
```

```
map[string]json.RawMessage{
```

```
compatibility := struct {  
    HomePage string  
    `json:"homePage"`  
}
```

```
if _, exists := fields["linkedIn"]; !exists &&  
strings.HasPrefix(compatibility.HomePage,  
linkedInPrefix) {  
    latest.LinkedIn =  
(compatibility.HomePage)[len(linkedInPrefix):]  
}
```

```
type speakerLatest struct {  
    Name      string `json:"name"`  
    Topic     string `json:"topic"`  
    LinkedIn  string `json:"linkedIn"`  
}
```



```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {
    ToDomainObject() (speaker, error)
}
```

```
type speakerData struct {
    Name      string `json:"name"`
    Topic     *string `json:"topic,omitempty"`
    HomePage *string
    `json:"homePage,omitempty"`
    LinkedIn  string `json:"linkedIn,omitempty"`
}
```

```
compatibility := struct {
    HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatibility)
json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

# Optional

Fast to change Null values Pointer errors

# Compatible

Small Functional Not clean

```
rest.Post("/v0.1.0/speaker", speakerCont0_1_0),  
rest.Post("/v0.2.0/speaker", speakerCont0_2_0),  
rest.Post("/v0.3.0/speaker", speakerCont0_3_0),
```

```
type speakerInputToDomain interface {  
    ToDomainObject() (speaker, error)  
}
```

```
type speakerData struct {  
    Name      string `json:"name"`  
    Topic     *string `json:"topic,omitempty"`  
    HomePage  *string  
    `json:"homePage,omitempty"`  
    LinkedIn  string `json:"linkedIn,omitempty"`  
}
```

```
compatibility := struct {  
    HomePage string `json:"homePage"`  
}{}  
fields := map[string]json.RawMessage{
```

```
json.Unmarshal(payload, &latest)  
json.Unmarshal(payload, &compatibility)  
json.Unmarshal(payload, &fields)
```

# URL

Clean Functional Code noise

# Optional

Fast to change Null values Pointer errors

# Compatible

Small Functional Not clean

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Logging

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Logging

# Input validation prevents unexpected behaviour

The problem – you can document endpoints, but

**POST** /user Create user

This can only be done by the logged in user.

Name

Description

**body** \* required

Created user object

```
User {  
  id integer(sint64)  
  first_name string  
  last_name string  
  email string  
  password string  
  phone string  
  userStatus integer(sint32)  
  User Status  
}
```

Your users might be really creative  
Also helps for integration testing

Responses

Response con

**PUT** /pet Update an existing pet

Parameters

Try it out

Name

Description

**body** \* required

Pet object that needs to be added to the store

(body)

Example ValueModel

```
{  
  "id": 0,  
  "category": {  
    "id": 0,  
    "name": "string"  
  },  
  "name": "doggie",  
  "photoUrls": [  
    "string"  
  ],  
  "tags": [  
    {  
      "id": "string"  
    },  
    {  
      "id": "string"  
    }  
  ],  
  "status": "available"  
}
```

Parameter content type

application/xml

Responses

Response content type

application/xml

Code

Description

400

Invalid ID supplied

404

Pet not found

405

Validation exception

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

# Custom parsers

# External library

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal **string** into Go struct field  
Data.location of type **uint64***

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
validate:"numeric,len=13"  
}
```

# Builtin parsers

## Custom parsers

## External library



```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal **string** into Go struct field  
Data.location of type **uint64***

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
{"number": 100}
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
type LocationNumber uint64
```

```
type LocationData struct {  
    Number LocationNumber `json:"number"`  
}
```

```
{"number": 100}
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
type LocationNumber uint64
```

```
type LocationData struct {  
    Number LocationNumber `json:"number"`  
}
```

```
{"number": 100}
```

```
{"number": 1.0e+8}
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
type LocationNumber uint64
```

```
type LocationData struct {  
    Number LocationNumber `json:"number"`  
}
```

```
{"number": 100}
```

```
{"number": 1.0e+8}
```

**cannot unmarshal** number 1.0e+8 into  
Go struct field LocationData.number of  
type main.LocationNumber

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
type LocationNumber uint64
```

```
type LocationData struct {  
    Number LocationNumber `json:"number"`  
}
```

```
{"number": 100}
```

```
{"number": 1.0e+8}
```

**cannot unmarshal number 1.0e+8 into  
Go struct field LocationData.number of  
type main.LocationNumber**

```
func (i *LocationNumber) UnmarshalJSON(data []byte) error {  
    var longInteger uint64  
    var longFloat float64
```

```
    errLong := json.Unmarshal(data, &longInteger)  
    errFloat := json.Unmarshal(data, &longFloat)
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal **string** into Go struct field  
Data.location of type **uint64***

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

## Custom parsers

## External library

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DateExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

# Custom parsers

# External library



```
type Data struct {
    Location uint64 `json:"location"`
}
```

```
err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DateExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

```
var parsedTime time.Time = time.Now()
err = json.Unmarshal([]byte(`"+parsed+"T00:00:00Z`), &parsedTime)
if err != nil {
    return errors.Annotatef(err, errorMessage, data)
}
*d = Date(parsedTime)
```



```
type Data struct {  
    Location uint64 `json:"location"`  
}
```

```
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DateExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

```
var parsedTime time.Time = time.Now()  
err = json.Unmarshal([]byte(`"+parsed+"T00:00:00Z"), &parsedTime)  
if err != nil {  
    return errors.Annotatef(err, errorMessage, data)  
}
```

```
*d = Date(parsedTime)
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}
```

```
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DateExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

# JSON has type Go does not

```
var parsedTime time.Time  
err := json.Unmarshal([]byte(`{"date":"2017-09-27"}`), &parsedTime)  
if err != nil {  
    return err  
}  
*d = Date(parsedTime)
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}
```

```
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error
```

```
...  
func (d Date) MarshalJSON() ([]byte, error)
```

```
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

```
var parsedTime time.Time  
err := json.Unmarshal([]byte(`{"date":"2017-09-27T10:00:00Z"}`), &parsedTime)  
if err != nil {  
    return errors.Errorf("error parsing date: %v", err)  
}  
*d = Date(parsedTime)
```

# JSON does not Go has type

```
type Data struct {
    Location uint64 `json:"location"`
}
```

```
err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

```
var parsedTime time.Time = time.Now()
err = json.Unmarshal([]byte(`"+parsed+"T00:00:00Z`), &parsedTime)
if err != nil {
    return errors.Annotatef(err, errorMessage, data)
}
*d = Date(parsedTime)
```

```
{
  "set": [
    "a",
    "b",
    "c"
  ]
}
```

sort.Strings(...)

No real type for sets in JSON  
In go map keys are  
randomised

Ended up validating map  
identifier as a primary key for  
structure, so JSON Diff tools  
could work as expected

map[string]string

```
{
  "set": {
    "a": "a",
    "b": "b",
    "c": "c"
  }
}
```

```
type Data struct {  
    Location uint64 `json:"location"`  
}
```

```
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

```
{"date":"2017-09-27"}
```

```
{"date":"2017 September 27th"}
```

It was all about  
good stuff  
of custom  
parsers

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

# Why not to validate everything in JSON parsers

```

type Data struct {
    Location uint64 `json:"location"`
}

err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)

```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```

type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}

```

```

import "gopkg.in/go-playground/validator.v9"

```

```

type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}

```

```

{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": [
    {
      "type": "meetup",
      "attributes": {
        "language": "Go",
        "date": "2017-09-27"
      }
    }
  ]
}

```



```

type Data struct {
    Location uint64 `json:"location"`
}

err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)

```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```

type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}

```

```

import "gopkg.in/go-playground/validator.v9"

```

```

type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}

```

```

{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": [
    {
      "type": "meetup",
      "attributes": {
        "language": "Go",
        "date": "2017-09-27"
      }
    }
  ]
}

```

```

type GroupedElements
map[string]Elements
type Elements []Element

```



```

type Data struct {
    Location uint64 `json:"location"`
}
err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)

```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```

type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}

```

```

import "gopkg.in/go-playground/validator.v9"

```

```

type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}

```

```

func (i *Element) UnmarshalJSON(data []byte) error {
    intermediate := struct {
        Type string `json:"type"`
        Attributes json.RawMessage `json:"attributes"`
    }{}
}

```

```

{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": [
    {
      "type": "meetup",
      "attributes": {
        "language": "Go",
        "date": "2017-09-27"
      }
    }
  ]
}

```

```

type GroupedElements
map[string]Elements
type Elements []Element

```

```

IAttribute interface {
    Type() string
}
Element struct {
    attribute IAttribute
}

```

```
type Data struct {
    Location uint64 `json:"location"`
}
err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}
```

```
func (i *Element) UnmarshalJSON(data []byte) error {
    intermediate := struct {
        Type string `json:"type"
        Attributes json.RawMessage `json:"attributes"
    }{}
```

```
{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": [
    {
      "type": "meetup",
      "attributes": {
        "language": "Go",
        "date": "2017-09-27"
      }
    }
  ]
}
```

```
type GroupedElements
map[string]Elements
type Elements []Element
```

```
IAtribute interface {
    Type() string
}
Element struct {
    attribute IAtribute
}
```

```

type Data struct {
    Location uint64 `json:"location"`
}

err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)

```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```

func (i *Element) UnmarshalJSON(data []byte) error {
    intermediate := struct {
        Type string `json:"type"`
        Attributes json.RawMessage `json:"attributes"`
    }{}

```

```

type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}

```

```

import "gopkg.in/go-playground/validator.v9"

```

```

type DataExample struct {
    Email string `json:"email"`
    validate:"email,isExampleDomain"
    Gln string `json:"gln"`
    validate:"numeric,len=13"
}

```

```

{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": null
}

```

```

type GroupedElements
map[string]Elements
type Elements []Element

```

```

IAttribute interface {
    Type() string
}

Element struct {
    attribute IAttribute
}

```

```
type Data struct {
    Location uint64 `json:"location"`
}
err := json.Unmarshal(
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

```
type Date time.Time
func (d *Date) UnmarshalJSON(data []byte) error
...
func (d Date) MarshalJSON() ([]byte, error)
...
type DateExample struct {
    Date Date `json:"date"`
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {
    Email string `json:"email"
    validate:"email,isExampleDomain"
    Gln string `json:"gln"
    validate:"numeric,len=13"
}
```

```
func (i *Element) UnmarshalJSON(data []byte) error {
    intermediate := struct {
        Type string `json:"type"`
        Attributes json.RawMessage `json:"attributes"`
    }{}
```

```
{
  "images": [
    {
      "type": "image",
      "attributes": {
        "path": "http://tny.im/ajb"
      }
    }
  ],
  "meetups": null
}
```

```
type GroupedElements
map[string]Elements
type Elements []Element
```

Slice is a pointer.  
No UnmarshalJSON called, no validation

```
"meetups":main.Elements(nil)
```

Also Go does not have proper constructors (private properties per package) – so hard to ensure valid state

Another use case – how to create test data for unit testing

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DateExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

# Custom parsers

Part of tooling Unexpected behavior

# External library

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

# Custom parsers

Part of tooling Unexpected behavior

# External library

# Layers of validation

```
type Data struct {  
    Location uint64 `json:"location"`  
}  
  
err := json.Unmarshal(  
    []byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field Data.location of type uint64*

## White list approach to unmarshal

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

## Black list approach to validate on structs

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
    validate:"numeric,len=13"  
}
```

## Database/state tests will also be a layer



```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
    []byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

# Custom parsers

Part of tooling Unexpected behavior

# External library

Accuracy Ecosystem Easy to forget to call



```
type Data struct {  
    Location uint64 `json:"location"`  
}  
err := json.Unmarshal(  
[]byte(`{"Location":"Vilnius"}`), &data)
```

*cannot unmarshal string into Go struct field  
Data.location of type uint64*

```
type Date time.Time  
func (d *Date) UnmarshalJSON(data []byte) error  
...  
func (d Date) MarshalJSON() ([]byte, error)  
...  
type DateExample struct {  
    Date Date `json:"date"`  
}
```

```
import "gopkg.in/go-playground/validator.v9"
```

```
type DataExample struct {  
    Email string `json:"email"`  
    validate:"email,isExampleDomain"  
    Gln string `json:"gln"`  
    validate:"numeric,len=13"  
}
```

# Builtin parsers

Fast Tested One not fits all

# Custom parsers

Part of tooling Unexpected behavior

# External library

Accuracy Ecosystem Easy to forget to call

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Logging

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Debugging

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
  "version": "v2.4"},  
{"version": "v3",  
  "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: IntPtr(2),  
        Episode: IntPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Type casting and update by reference

## Logging actual values instead of pointers

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Acceptance tests

Type casting and  
update by reference

Logging actual values  
instead of pointers

```
Expect(response).Should(
    MatchJSON(expected)
)
```

```
{"title": "Google Analytics API",
 "version": "v2.4"},
{"version": "v3",
 "title": "Google Analytics API"},
```

```
meetups := Meetups{
    Meetup{
        Name: "Golang Meetup
S02E01",
        Session: intPtr(2),
        Episode: intPtr(1),
        Speakers: []Speaker{
            {
                "Aurelijus Banelis",
```

```
meetups := Meetups{
    input := `[
        {
            "name": "Golang Meetup
S02E01",
            "session": 2,
            "episode": 1,
            "Speakers": [
                {
                    "name": "Aurelijus
Banelis",
```

```
representation := fmt.Sprintf("%#v",
meetups)
```

```
test.Meetups{test.Meetup{Session:(*
uint)(0xc42028c5d0),
Episode:(*uint)(0xc42028c5d8) ....
```

[https://www.googleapis.com/discovery/v1/apis?name=analytics&fields=items\(description%2CdocumentationLink%2Ctitle%2Cversion\)](https://www.googleapis.com/discovery/v1/apis?name=analytics&fields=items(description%2CdocumentationLink%2Ctitle%2Cversion))

```
▼ items:
  ▼ 0:
    version:      "v2.4"
    title:        "Google Analytics API"
    description:  "Views and manages your Google Analytics data."
    documentationLink: "https://developers.google.com/analytics/"
  ▼ 1:
    version:      "v3"
    title:        "Google Analytics API"
    description:  "Views and manages your Google Analytics data."
    documentationLink: "https://developers.google.com/analytics/"
```

Easier to recheck results and paste rather than writing everything word by word (TDD)

```
Expect(response).Should(
    MatchJSON(expected)
)
```

```
{"title": "Google Analytics API",
 "version": "v2.4"},
{"version": "v3",
 "title": "Google Analytics API"},
```

```
meetups := Meetups{
    Meetup{
        Name: "Golang Meetup
S02E01",
        Session: intPtr(2),
        Episode: intPtr(1),
        Speakers: []Speaker{
            {
                "Aurelijus Banelis",
```

```
meetups := Meetups{
    input := `[
        {
            "name": "Golang Meetup
S02E01",
            "session": 2,
            "episode": 1,
            "Speakers": [
                {
                    "name": "Aurelijus
Banelis",
```

```
representation := fmt.Sprintf("%#v",
meetups)
```

```
test.Meetups{test.Meetup{Session:(*
uint)(0xc42028c5d0),
Episode:(*uint)(0xc42028c5d8) ....
```

[https://www.googleapis.com/discovery/v1/apis?name=analytics&fields=items\(description%2CdocumentationLink%2Ctitle%2Cversion\)](https://www.googleapis.com/discovery/v1/apis?name=analytics&fields=items(description%2CdocumentationLink%2Ctitle%2Cversion))

```
▼ items:
  ▼ 0:
    version: "v2.4"
    title: "Google Analytics API"
    description: "Views and manages your Google Analytics data."
    documentationLink: "https://developers.google.com/analytics/"
  ▼ 1:
    version: "v3"
    title: "Google Analytics API"
    description: "Views and manages your Google Analytics data."
    documentationLink: "https://developers.google.com/analytics/"
```

Easier to recheck results and paste rather than writing everything word by word (TDD)  
Also easier when communicating



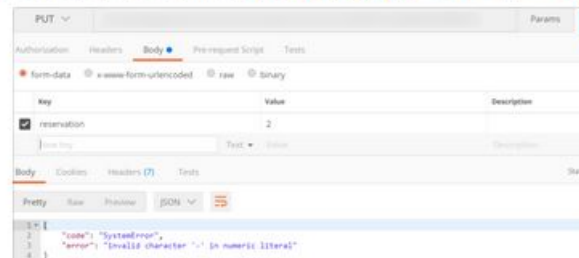
3:29 PM

can you help me understand why my request is failing, here is the details



3:29 PM

uploaded this image: [Pasted image at 3:29 PM](#)





```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{  
  "title": "Google Analytics API",  
  "version": "v2.4"},  
{  
  "version": "v3",  
  "title": "Google Analytics API",  
}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Type casting and update by reference

## Logging actual values instead of pointers



```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Type casting and update by reference

## Logging actual values instead of pointers

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Unit tests

# Logging actual values instead of pointers

```
Expect(response).Should(
    MatchJSON(expected)
)
```

```
{ "title": "Google Analytics API",
  "version": "v2.4"},
{ "version": "v3",
  "title": "Google Analytics API"},
```

```
meetups := Meetups{
    Meetup{
        Name: "Golang Meetup
S02E01",
        Session: IntPtr(2),
        Episode: IntPtr(1),
        Speakers: []Speaker{
            {
                "Aurelijus Banelis",
```

```
meetups := Meetups{
input := `[
    {
        "name": "Golang Meetup
S02E01",
        "session": 2,
        "episode": 1,
        "speakers": [
            {
                "name": "Aurelijus
Banelis",
```

```
representation := fmt.Sprintf("%#v",
meetups)
```

```
test.Meetups{test.Meetup{Session:(*
uint)(0xc42028c5d0),
Episode:(*uint)(0xc42028c5d8) ....
```

```
Meetup{
    Name: "Golang Meetup S02E01",
    Session: IntPtr(2),
    Episode: IntPtr(1),
    Speakers: []Speaker{
        {
            FirstLastName("Aurelijus Banelis"),
            Topic{
                "JSON+Go in practice",
                HTML("Practical tips and tricks ..."),
            },
            &Company{
                "NFQ",
                Url("http://www.nfq.it/"),
            },
        },
    },
}
{
    FirstLastName("Valdas Petrulis"),
    Topic{
        "NATS pub-sub daemon packed inside your application ",
        HTML("Here at www.mysterium.network ..."),
    },
    &Company{
        "MysteriumNetwork",
        Url("http://www.mysterium.network/"),
    },
},
},
Date: Date("2017-09-27"),
Description: HTML("Arriving: Call Povilas if you'll face any problems reaching
the place."),
Sponsor: Company{
    "Uber",
    Url("https://www.uber.com/en-LT/"),
},
Venue: Place{
    Name: "Uber",
    Address: Address("Lvovo g. 25, Vilnius"),
},
},
```

```
{
    "name": "Golang Meetup S02E01",
    "session": 2,
    "episode": 1,
    "speakers": [
        {
            "name": "Aurelijus Banelis",
            "topic": {
                "title": "JSON+Go in practice",
                "description": "Practical tips and tricks ..."
            },
            "company": {
                "name": "NFQ",
                "url": "http://www.nfq.it/"
            }
        },
    ],
    "date": "2017-09-27",
    "venue": {
        "name": "Uber",
        "address": "Lvovo g. 25, Vilnius"
    },
    "sponsor": {
        "name": "Uber",
        "url": "https://www.uber.com/en-LT/"
    },
    "description": "Arriving: Call Povilas if you'll face any problems
reaching the place."
}
```

Sometimes it is shorter to write JSON than all pointer, casting and inner structure (especially with older Go)

```
Expect(response).Should(
    MatchJSON(expected)
)
```

```
{ "title": "Google Analytics API",
  "version": "v2.4"},
{ "version": "v3",
  "title": "Google Analytics API"},
```

```
meetups := Meetups{
    Meetup{
        Name: "Golang Meetup
S02E01",
        Session: intPtr(2),
        Episode: intPtr(1),
        Speakers: []Speaker{
            {
                "Aurelijus Banelis",
```

```
meetups := Meetups{
    input := `[
        {
            "name": "Golang Meetup
S02E01",
            "session": 2,
            "episode": 1,
            "speakers": [
                {
                    "name": "Aurelijus
Banelis",
```

```
representation := fmt.Sprintf("%#v",
    meetups)
```

```
test.Meetups{test.Meetup{Session:(*
uint)(0xc42028c5d0),
Episode:(*uint)(0xc42028c5d8) ....
```

```
Meetup{
    Name: "Golang Meetup S02E01",
    Session: intPtr(2),
    Episode: intPtr(1),
    Speakers: []Speaker{
        {
            FirstLastName("Aurelijus Banelis"),
            Topic{
                "JSON+Go in practice",
                HTML("Practical tips and tricks ..."),
            },
            &Company{
                "NFQ",
                Url("http://www.nfq.it/"),
            },
        },
    },
    {
        FirstLastName("Valdas Petrulis"),
        Topic{
            "NATS pub-sub daemon packed inside your application ",
            HTML("Here at www.mysterium.network ..."),
        },
        &Company{
            "MysteriumNetwork",
            Url("http://www.mysterium.network/"),
        },
    },
},
    Date: Date("2017-09-27"),
    Description: HTML("Arriving: Call Povilas if you'll face any problems reaching
the place."),
    Sponsor: Company{
        "Uber",
        Url("https://www.uber.com/en-LT"),
    },
    Venue: Place{
        Name: "Uber",
        Address: Address("Lvovo g. 25, Vilnius"),
    },
},
```

```
{
    "name": "Golang Meetup S02E01",
    "session": 2,
    "episode": 1,
    "speakers": [
        {
            "name": "Aurelijus Banelis",
            "topic": {
                "title": "JSON+Go in practice",
                "description": "Practical tips and tricks ..."
            },
            "company": {
                "name": "NFQ",
                "url": "http://www.nfq.it/"
            }
        },
        {
            "name": "Valdas Petrulis",
            "topic": {
                "title": "NATS pub-sub daemon packed inside your application ",
                "description": "Here at www.mysterium.network ..."
            },
            "company": {
                "name": "MysteriumNetwork",
                "url": "http://www.mysterium.network/"
            }
        }
    ],
    "date": "2017-09-27",
    "venue": {
        "name": "Uber",
        "address": "Lvovo g. 25, Vilnius"
    },
    "sponsor": {
        "name": "Uber",
        "url": "https://www.uber.com/en-LT/"
    },
    "description": "Arriving: Call Povilas if you'll face any problems
reaching the place."
}
```

Sometimes it is shorter to write JSON than all pointer, casting and inner structure (especially with older Go)  
Also JSON unmarshal prevents accidental update by reference

```
actual := map[string]*string{}
expected := map[string]*string{}
```

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Type casting and update by reference

## Logging actual values instead of pointers

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

Copy example directly  
from logs or curl

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

Type casting and  
update by reference

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

Logging actual values  
instead of pointers

```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# Copy example directly from logs or curl

## Type casting and update by reference

# Debugging



```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API"},
```

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
        {  
            "name": "Golang Meetup  
S02E01",  
            "session": 2,  
            "episode": 1,  
            "Speakers": [  
                {  
                    "name": "Aurelijus  
Banelis",
```

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

# From the days Go had bad debugger support...

## JSON Diff

The semantic JSON compare tool

Found 14 differences

Show: ☒ 4 missing properties ☒ 10 unequal values

```
1. {  
2.   "date": "2017-09-27",  
3.   "description": "Arriving: Call Povilas if you'll face any problems reaching the place.",  
4.   "episode": 1,  
5.   "name": "Golang Meetup S02E01",  
6.   "session": 2,  
7.   "Speakers": [  
8.     {  
9.       "company": {  
10.        "name": "HBO",  
11.        "url": "http://www.nfg.lt/"  
12.      },  
13.      "name": "Aurelijus Banelis",  
14.      "topic": {  
15.        "description": "Practical tips and tricks ...",  
16.        "title": "JSON+Go in practice"  
17.      },  
18.    },  
19.  ],  
20.   "company": {  
21.     "name": "MysteriumNetwork",  
22.     "url": "http://www.mysterium.network/"  
23.   },  
24.   "name": "Valdas Petrusis",  
25.   "topic": {  
26.     "description": "Here at www.mysterium.network ...",  
27.     "title": "MATS pub-sub daemon packed inside your application"  
28.   },  
29. },  
30. "sponsor": {  
31.   "name": "Uber",  
32.   "url": "https://www.uber.com/en-LT/"  
33. },  
34. "venue": {  
35.   "address": "Lvovo g. 25, Vilnius",  
36.   "name": "Uber"  
37. },  
38. },  
39. },  
40. },  
41. }
```

Validate, format, and compare two JSON documents. See the differences between the objects instead of just the new lines and mixed up properties.

Created by [Zack Grossbart](#). Get the [source code](#).

Big thanks owed to the team behind [JSONLint](#).

Perform a new diff

```
1. {  
2.   "date": "2017-09-24",  
3.   "description": "Arriving: Call Povilas if you'll face any problems reaching the place.",  
4.   "name": "Golang Meetup #5",  
5.   "Speakers": [  
6.     {  
7.       "company": {  
8.        "name": "Weaveworks",  
9.        "url": "https://www.weave.works/"  
10.      },  
11.      "name": "Martynas Pumputis",  
12.      "topic": {  
13.        "description": "This talk is about discovering Go runtime limitations while  
14.        "title": "Go and Linux Namespaces: Better Don't Mix"  
15.      },  
16.    },  
17.    {  
18.      "name": "Mike Kabischev",  
19.      "topic": {  
20.        "description": "Instrumentation ...",  
21.        "title": "Instrumenting Go application"  
22.      },  
23.    },  
24.  ],  
25.   "name": "Max Chechel",  
26.   "topic": {  
27.     "description": "Why do we need to generate Go ...",  
28.     "title": "Code generation in go"  
29.   },  
30. },  
31. },  
32. "sponsor": {  
33.   "name": "Uber",  
34.   "url": "https://www.uber.com/en-LT/"  
35. },  
36. "venue": {  
37.   "address": "Lvovo g. 25, Vilnius",  
38.   "name": "Uber"  
39. },  
40. },  
41. }
```

< 1 of 14 >

Both sides should be  
equal strings

More tooling for JSON, than `fmt.Printf`  
But not always work, if properties intended to be private



```
Expect(response).Should(  
    MatchJSON(expected)  
)
```

```
{"title": "Google Analytics API",  
 "version": "v2.4"},  
{"version": "v3",  
 "title": "Google Analytics API",}
```

Copy example directly  
from logs or curl

```
meetups := Meetups{  
    Meetup{  
        Name: "Golang Meetup  
S02E01",  
        Session: intPtr(2),  
        Episode: intPtr(1),  
        Speakers: []Speaker{  
            {  
                "Aurelijus Banelis",
```

```
meetups := Meetups{  
    input := `[  
    {  
        "name": "Golang Meetup  
S02E01",  
        "session": 2,  
        "episode": 1,  
        "Speakers": [  
            {  
                "name": "Aurelijus  
Banelis",
```

Type casting and  
update by reference

```
representation := fmt.Sprintf("%#v",  
meetups)
```

```
test.Meetups{test.Meetup{Session:(*  
uint)(0xc42028c5d0),  
Episode:(*uint)(0xc42028c5d8) ....
```

Logging actual values  
instead of pointers

# Communication

Naming

Versioning

Validating

# Development

Acceptance tests

Unit tests

Debugging

# All working code examples

<https://gist.github.com/aurelijusb/5640816026e8b5f0e979a26be1e79ac2>

# Slides

<https://aurelijus.banelis.lt/presentations/golang-vilnius-2017/json-go-in-practice-v1.pdf>

JSON+Go in practice

# Questions?

Aurelijus Banelis

