JSON+Go in practice

Aurelijus Banelis

Aurelijus Banelis

Software developer

aurelijus.banelis.lt
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

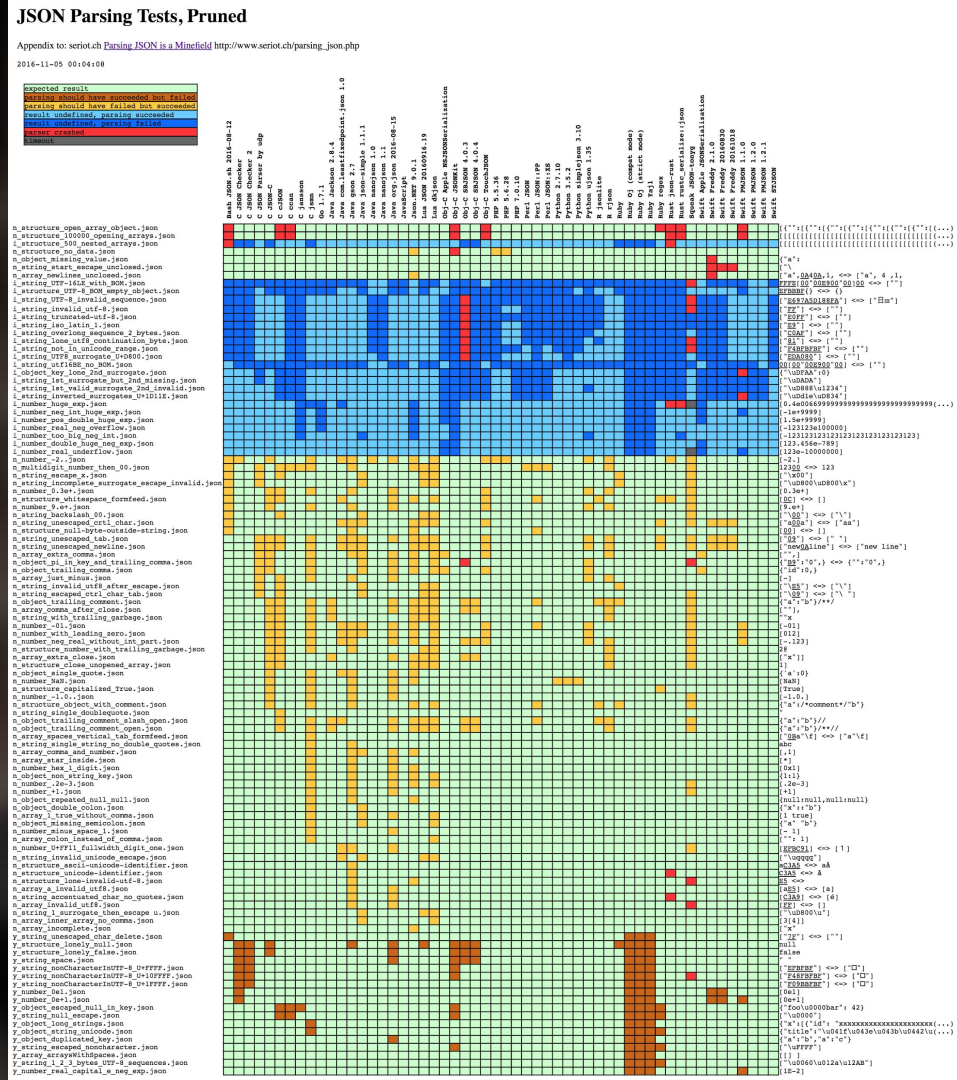
https://play.golang.org/p/N2ba_mgHKC

RFC 7159

The JavaScript Object Notation (JSON) Data Interchange Format

leaves many details loosely specified

http://seriot.ch/parsing_json.php



Communication

Naming

Versioning

Validating

Development

Acceptance tests

Unit tests

Logging

Communication

Naming

Versioning

Validating

Development

Acceptance tests

Unit tests

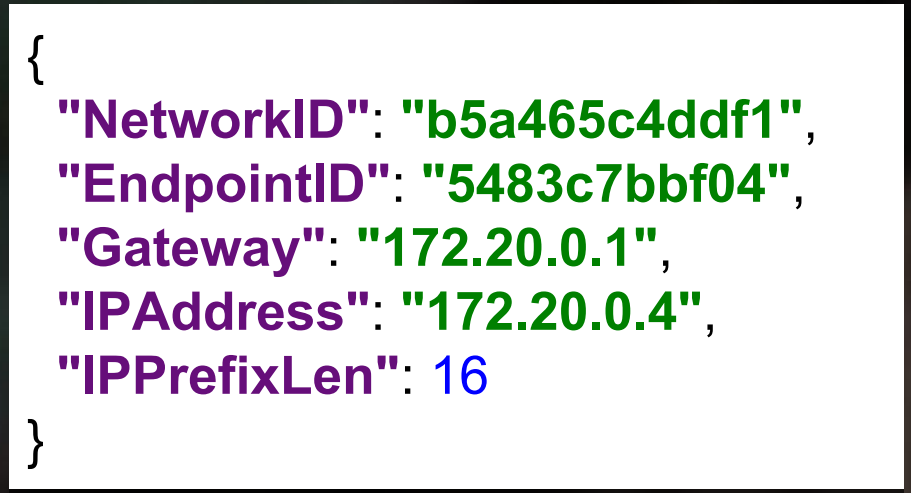
Logging

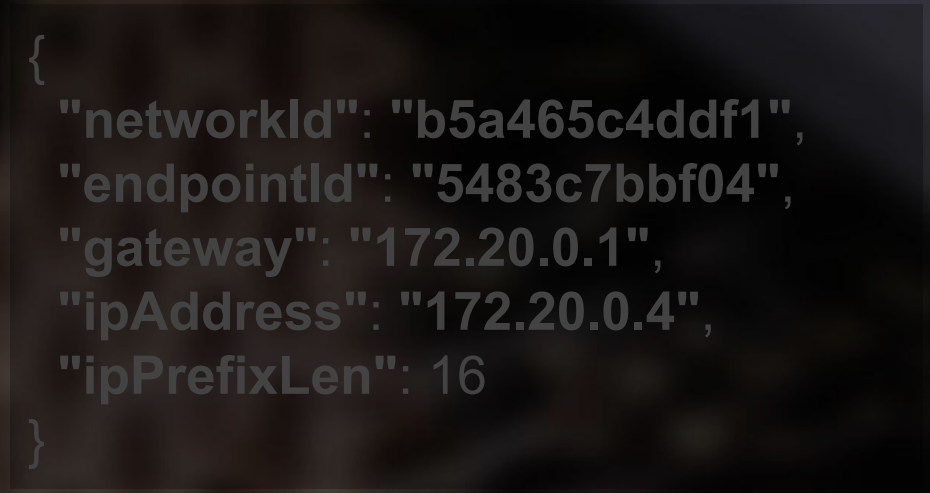
```
{
  "NetworkID": "b5a465c4ddf1",
  "EndpointID": "5483c7bbf04",
  "Gateway": "172.20.0.1",
  "IPAddress": "172.20.0.4",
  "IPPrefixLen": 16
}
```

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

Problem: I like to name it this way

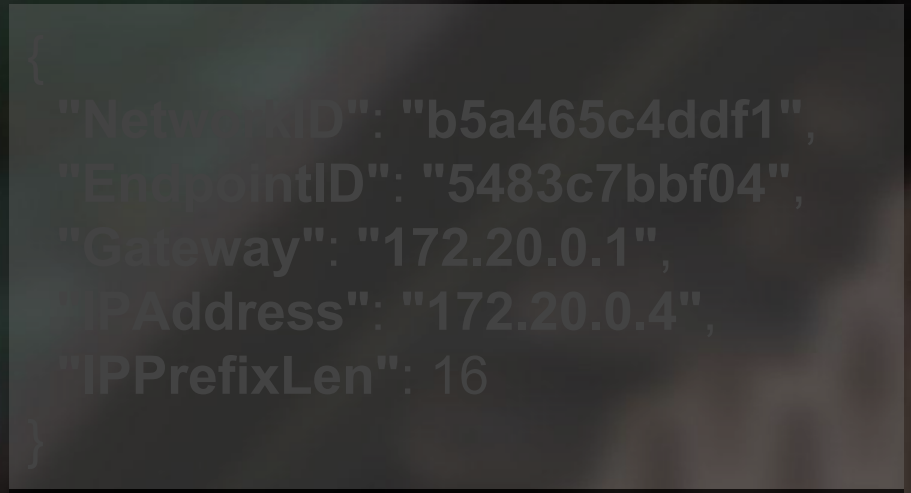
JSON is all about communication
Which part is written by Go and
not-Go programmer





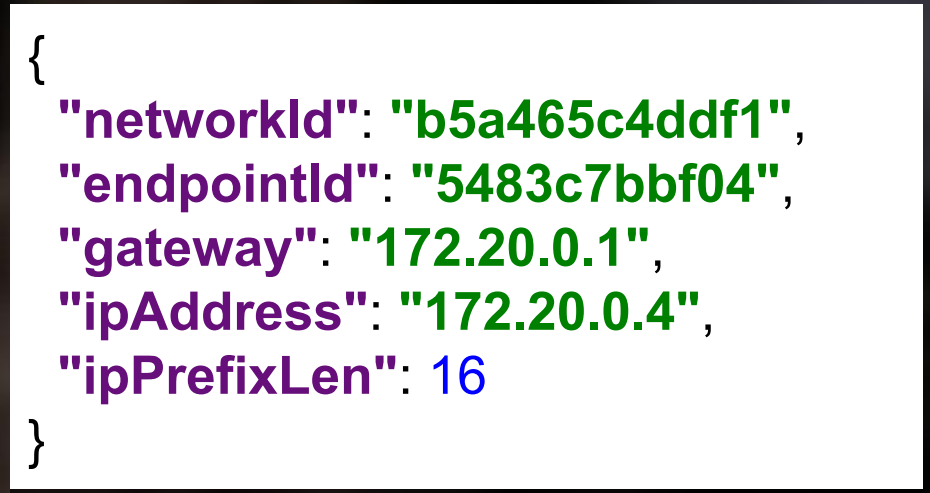
go lint

```
struct field
NetworkId should
be NetworkID
```



go lint

`struct field NetworkId should be NetworkID`



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
}
```

```go
import "encoding/json"

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

https://play.golang.org/p/UEMKMl6tC2

```go
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/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"`
}
```

Manual

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

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

Dynamic

```
// 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"`
}
```

Generated

```go
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"`
}
```

Manual

Well known Boring

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

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

Dynamic

```go
// 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"`
}
```

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"`
}
```

Manual

Well known Boring

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

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

Dynamic

Runtime errors Automated

```
// 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"`
}
```

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"`
}
```

Manual

Well known Boring

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

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

Dynamic

Runtime errors Automated

```
// 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"`
}
```

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"
    }
  }
]
```

Polymorphism

```
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"
    }
  }
]
```

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

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

```go
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"`
}
```

Manual

Well known Boring

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

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

Dynamic

Runtime errors Automated

```go
// 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"`
}
```

Generated

Compile time check Learning curve

Communication

Naming

Versioning

Validating

Development

Acceptance tests

Unit tests

Logging

Communication

Naming

Versioning

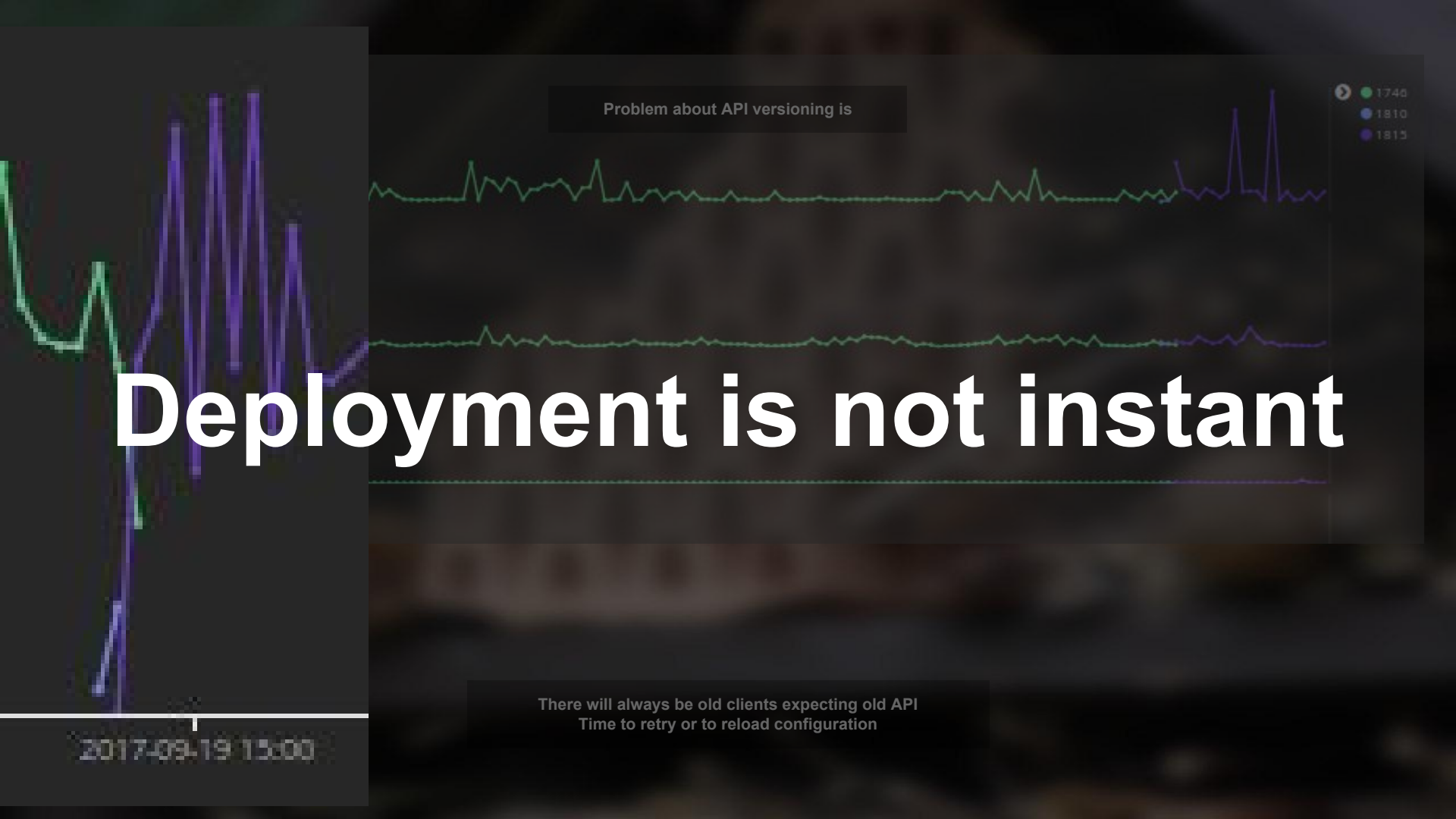
Validating

Development

Acceptance tests

Unit tests

Logging



Deployment is not instant

```
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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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)
}
```

URL

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

Optional

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

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

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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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)
}
```

URL

Clean Functional Code noise

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

Optional

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

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

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)
}
```

URL

Clean Functional Code noise

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

Optional

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

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

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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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"`
}
```

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

json.Unmarshal(payload, &latest)
json.Unmarshal(payload, &compatiblity)
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)
}
```

URL

Clean Functional Code noise

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

Optional

Fast to change Null values Pointer errors

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

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

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)
}
```

URL

Clean Functional Code noise

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

Optional

Fast to change Null values Pointer errors

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

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

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"`
}
```

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

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

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

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

```
compatiblity := 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"`
}
```

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

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

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

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

```
compatiblity := struct {
  HomePage string
    `json:"homePage"`
```

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

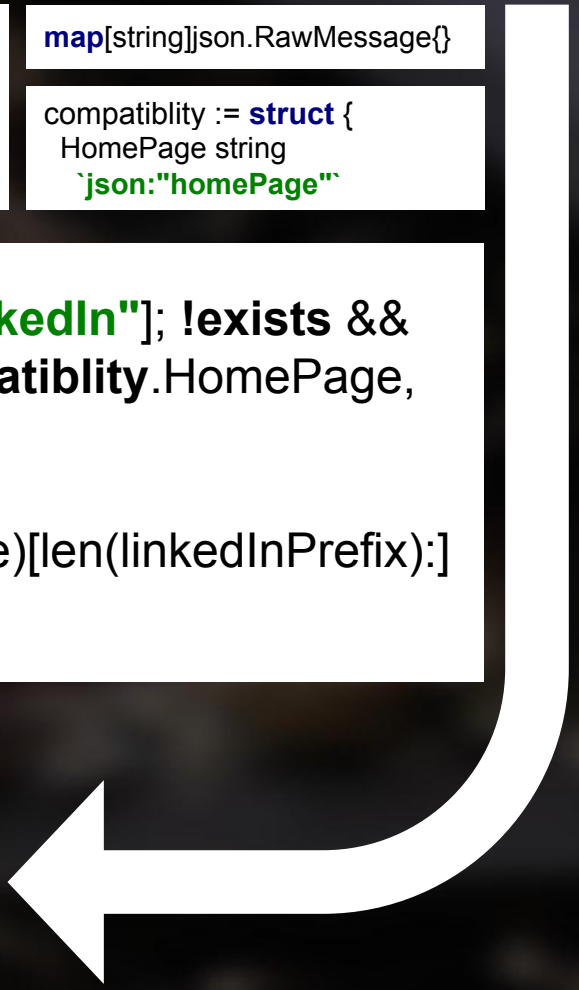
```
compatiblity := struct {
  HomePage string `json:"homePage"`
}{}
fields := map[string]json.RawMessage{}

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

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

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

```
compatiblity := struct {
  HomePage string
   `json:"homePage"`
```

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

URL

Clean Functional Code noise

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

Optional

Fast to change Null values Pointer errors

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

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

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)
}
```

URL

Clean Functional Code noise

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

Optional

Fast to change Null values Pointer errors

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

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

Compatible

Small Functional Not clean

Communication

Naming

Versioning

Validating

Development

Acceptance tests

Unit tests

Logging

Communication

Naming

Versioning

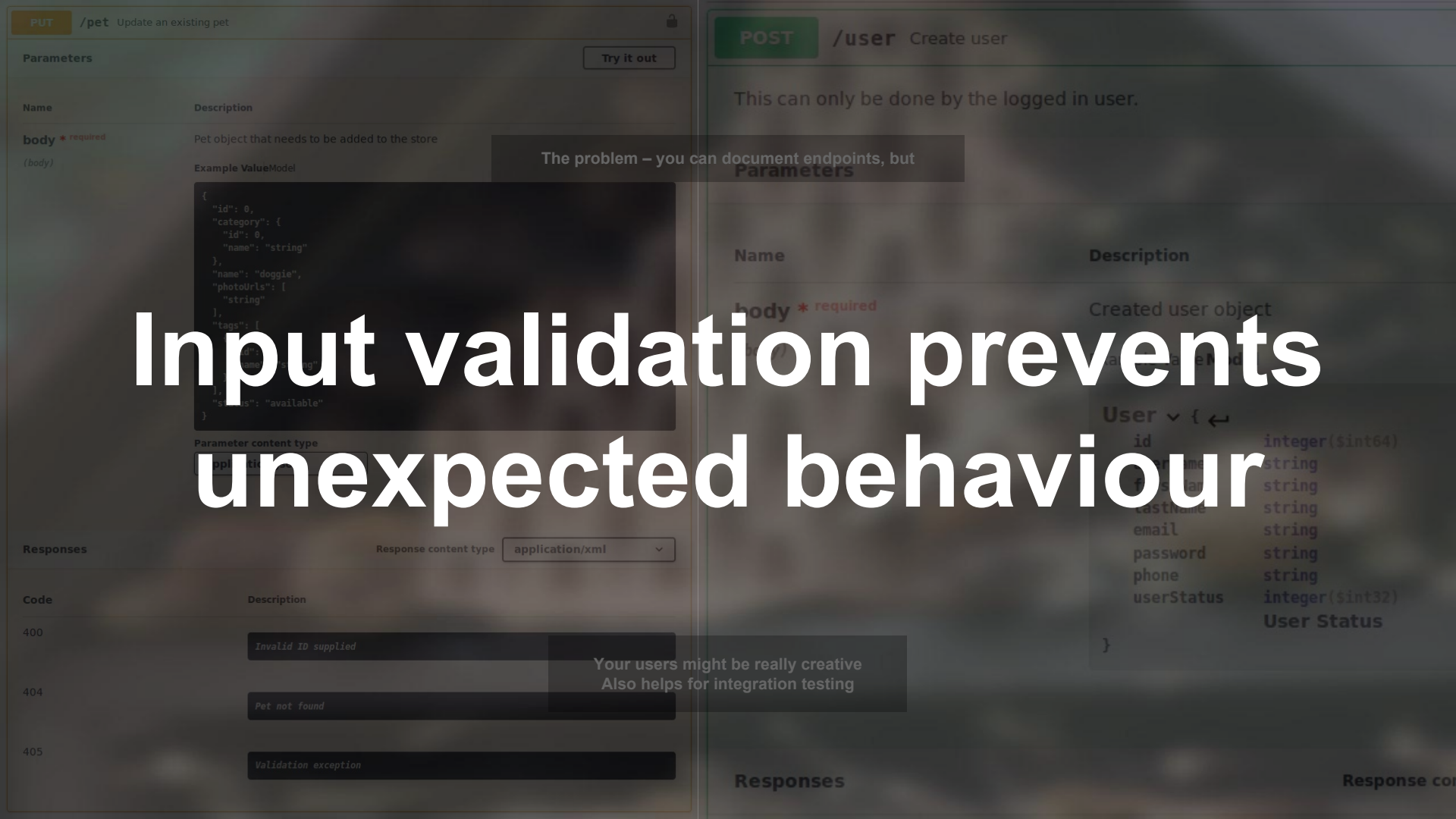
Validating

Development

Acceptance tests

Unit tests

Logging



Input validation prevents unexpected behaviour

```
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***

Builtin parsers

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

Custom parsers

```
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"`
}
```

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***

Builtin parsers

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

Custom parsers

```
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"`
}
```

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}
```

```go
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***

```go
type LocationNumber uint64
type LocationData struct {
  Number LocationNumber `json:"number"`
}
```

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

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

```go
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 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 LocationNumber uint64
type LocationData struct {
  Number LocationNumber `json:"number"`
}
```

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

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

```
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 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 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 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 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***

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

```
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"`
}
```

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***

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

```
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"`
}
```

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"`
}
```

```
{"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 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)
```

```
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
```

JSON has type Go does not

```
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
if err != nil {
  return
}
*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(`"`+parsed+"T00:00:00Z"`), &parsedTime)
if err != nil {
  return errors.Annotate(err, errorMessage, data)
}
*d = Date(parsedTime)
```

JSON has type Go does not

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"}

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

It was all about good stuff of custom parsers

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

sort.Strings(...)

In go map keys are randomised

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

map[string]string

```
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***

```
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"`
}
```

```
{
 "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
}
```

```
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 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"`
}
```

```
{
 "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
}
```

```
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 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
```

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

Part of tooling Unexpected behavior

```
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"`
}
```

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***

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

Part of tooling Unexpected behavior

```
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"`
}
```

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"`
}
```

White list approach to unmarshal

Black list approach to validate on structs

Database/state tests will also be a layer

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
```

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

Part of tooling Unexpected behavior

```
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"`
}
```

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***

Builtin parsers

Fast Tested One not fits all

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

Custom parsers

Part of tooling Unexpected behavior

```
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"`
}
```

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",}
```

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",}
```

Acceptance tests

```
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**) ....*

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",
```

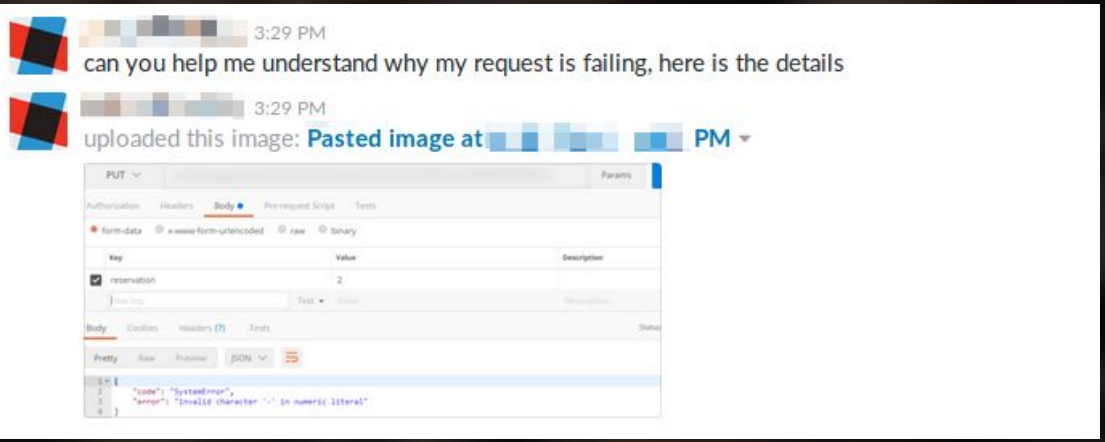
```
representation := fmt.Sprintf("%#v", meetups)
```

*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)


```
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



```
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",}
```

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",}
```

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",
```

Unit tests

```
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) ....
```

```
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.lt/"),
      },
    },
    {
      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.lt/"
  }
 },
 {
  "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)

```
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.lt/"),
      },
    },
    {
      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.lt/"
  }
 },
 {
  "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",}
```

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",}
```

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",}
```

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) ....
```

Debugging

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

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

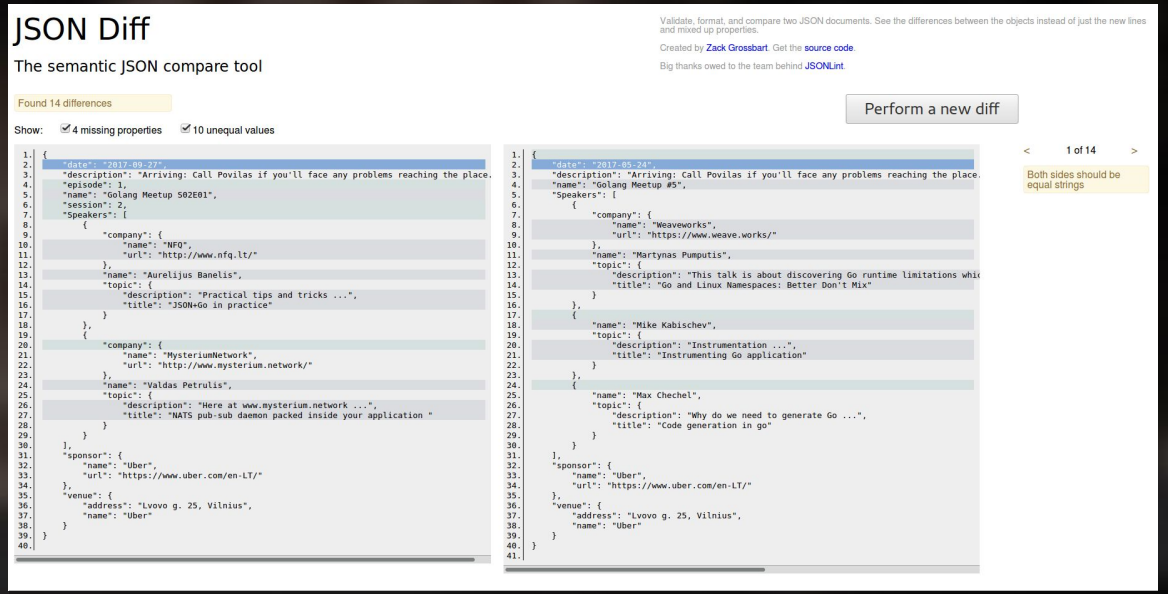
```
meetups := Meetups{
  Meetup{
    Name: "Golang Meetup S02E01",
    Session: intPtr(2),
    Episode: intPtr(1),
    Speakers:[]Speaker{
      {
        "Aurelijus Banelis",
```

```
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...



More tooling for JSON, than fmt #%v
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**/prezentations/golang-vilnius-2017/json-go-in-practice-v1.pdf

JSON+Go in practice

Questions?

Aurelijus Banelis