Write Once, Use Many

The Power of Encapsulation – Creating a Simple Reusable HTTP Package to Call Internal APIs

Michael Richman - Bitly

bitly

Who am I?

Michael Richman

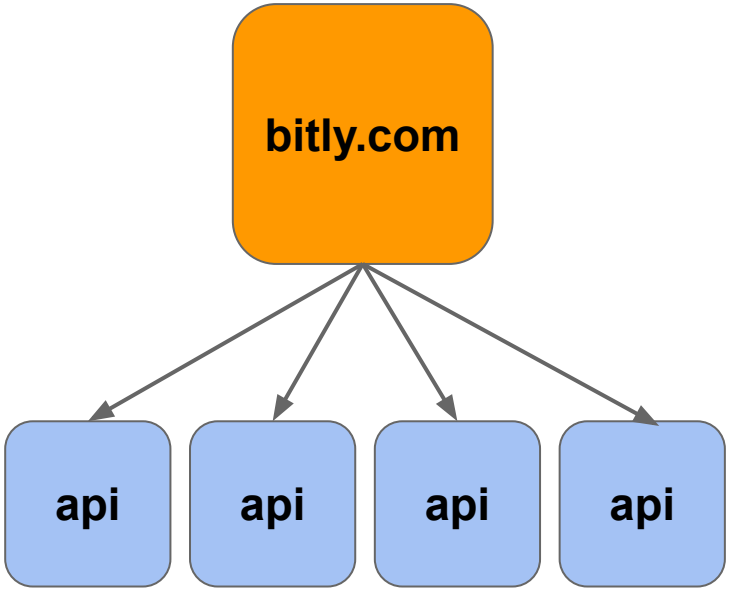
Senior Tech Lead - Bitly @mrwoofster

Bitly for 10 years
Software for 20 years
Lives in Denver

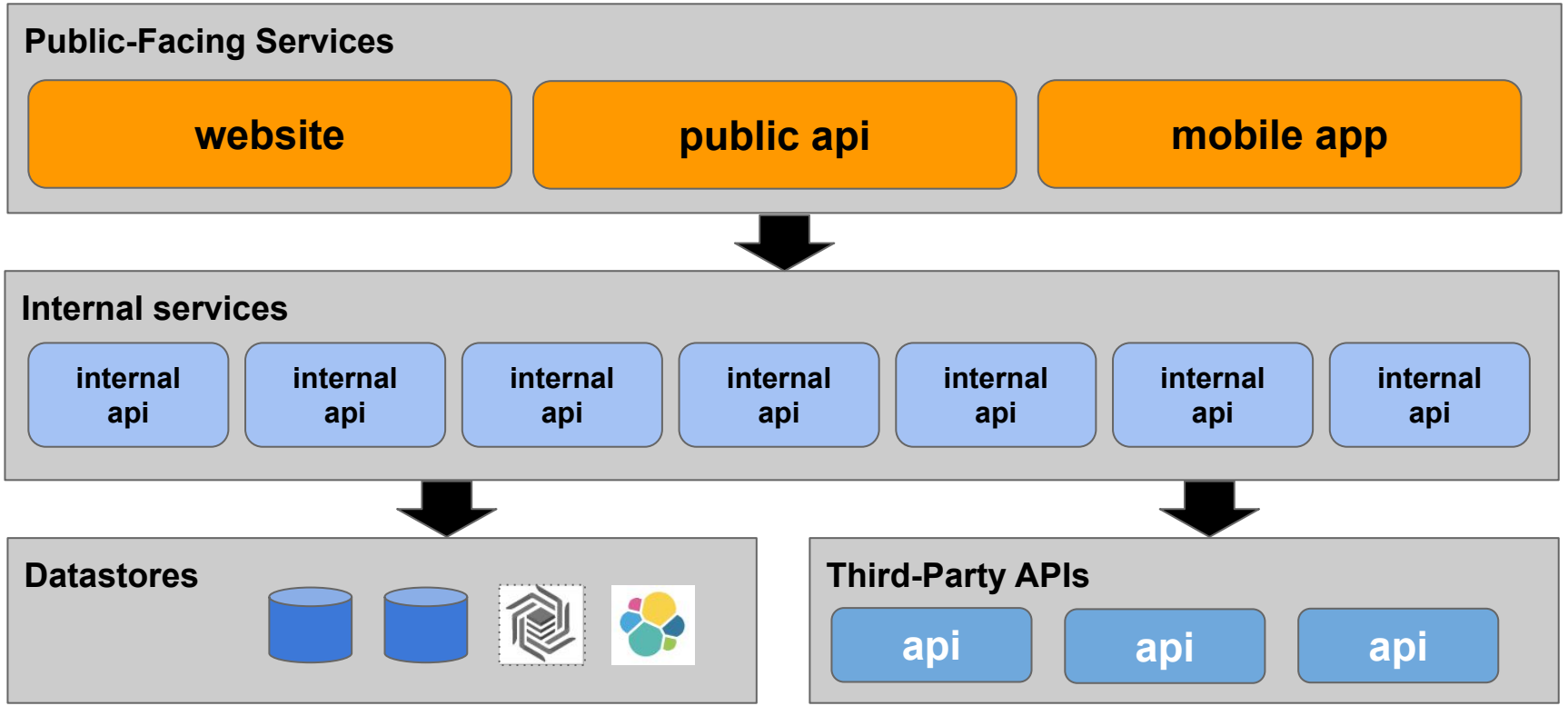
Bitly • bitly.com

Short links & link management
All in on Go since 2015

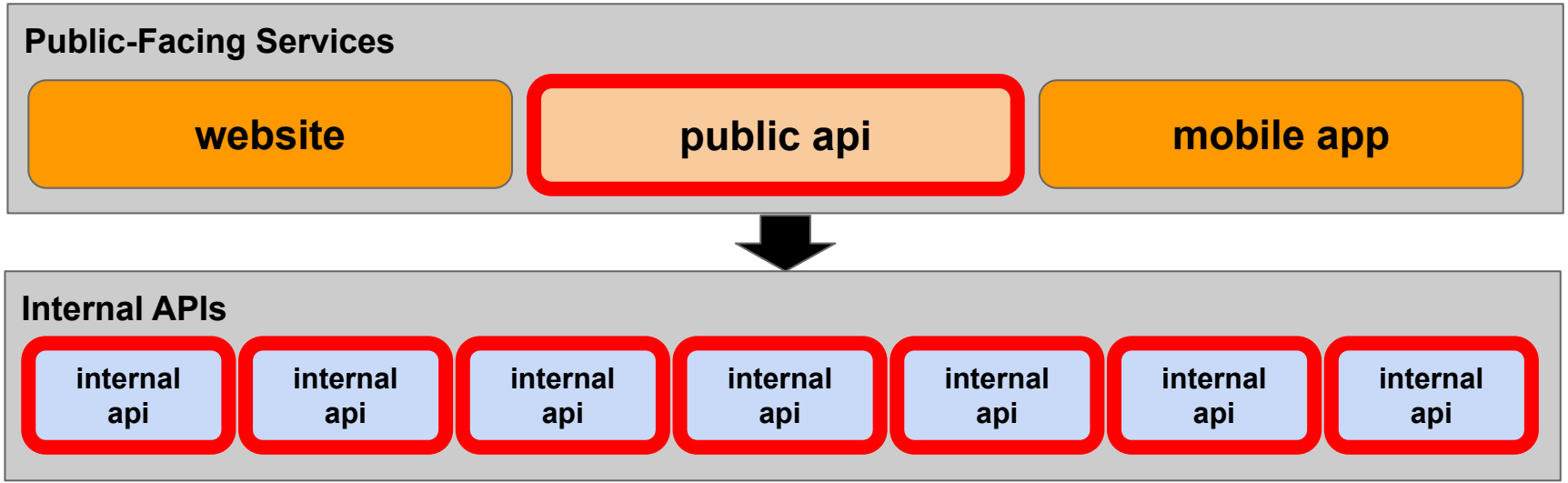
Like many of you, we make a lot of HTTP calls...



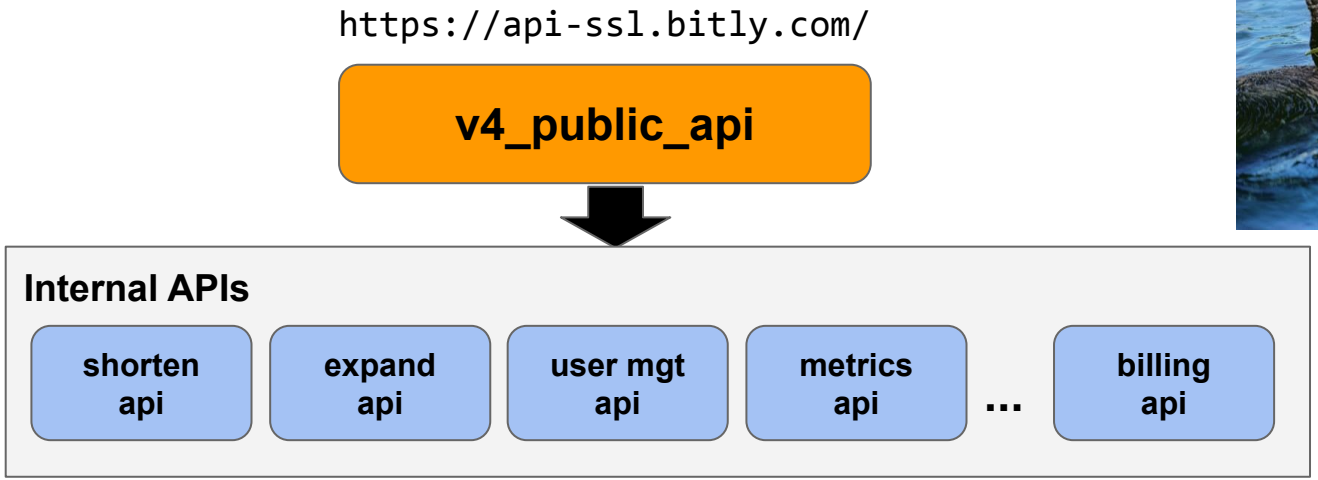
Bitly System Architecture



Our Focus Today: Bitly services calling other Bitly services



Looks something like this...



Let's say we want to write the code to call our internal APIs

We want to make some http calls

Standard lib HTTP calls

Google "golang http call" - `https://golang.org/pkg/net/http/`

Overview ▾

Package http provides HTTP client and server implementations.

Get, Head, Post, and PostForm make HTTP (or HTTPS) requests:

```
resp, err := http.Get("http://example.com/")
...
resp, err := http.Post("http://example.com/upload", "image/jpeg", &buf)
...
resp, err := http.PostForm("http://example.com/form",
        url.Values{"key": {"Value"}, "id": {"123"}})
```

Standard lib HTTP calls

https://golang.org/pkg/net/http/

For control over HTTP client headers, redirect policy, and other settings, create a Client:

```
client := &http.Client{
        CheckRedirect: redirectPolicyFunc,
}

req, err := http.NewRequest("GET", "http://example.com", nil)
// ...
req.Header.Add("If-None-Match", `W/"wyzzy"`)
resp, err := client.Do(req)
// ...
```

So let me write my code now

```
$ curl -XGET http://127.0.0.1:7000/orgs/Oi7nib6hJsY # API endpoint to get an Org

{"guid":"Oi7nib6hJsY","name":"mrwoof","tier_name":"enterprise","tier_family":"enterprise","tier_display
_name":"Enterprise","created_ts":1532370040,"modified_ts":1532370257,"activated_ts":1532370040,"deactiv
ated_ts":0,"is_active":true,"bsds":["mr.bitly.pro"],"status":"new","personal":""}
```

```
func GetOrg(orgGUID string) *usermanagementapi.Org {
    client := &http.Client{}
    req, err := http.NewRequest("GET", "https://usermanagementapi01.bitly.com/orgs/
    req.Header.Add("X-Bitly-User-Agent", "v4_public_api")
    resp, err := client.Do(req)
    b, err := ioutil.ReadAll(resp.Body)
    var org *usermanagementapi.Org
    err = jsonsafe.Unmarshal(b, &org)
    return org
}
```

Copy to...

```
GetUser()
GetGroup()
GetRoles()
GetCustomDomains()
Get...EVERYTHING()

Create...()
Update...()
Delete...()
```

But there's a problem. What if I want to add...

```
log.WithFields(log.Fields{"url": r.URL, "status_code": resp.StatusCode,
"duration_ms": reqDuration)).Info("request complete")
```

Consistent request logging

```
// service discov
host := pickHost(                                    ntapi02"})
port := findPortF
url := fmt.Sprint                                    t, port)
```

Service discovery

```
req.Header.Add("X
```

Universal headers

```
if resp.StatusCode >= 400 {...} // error handling...

if err := jsonsafe.Unmarshal(b, v); err != nil {...} // JSON

ReportToSentry(ctx, err)

if hook != nil {// call the hook instead of the real call}
```

Consistent error handling

Repeat everywhere!

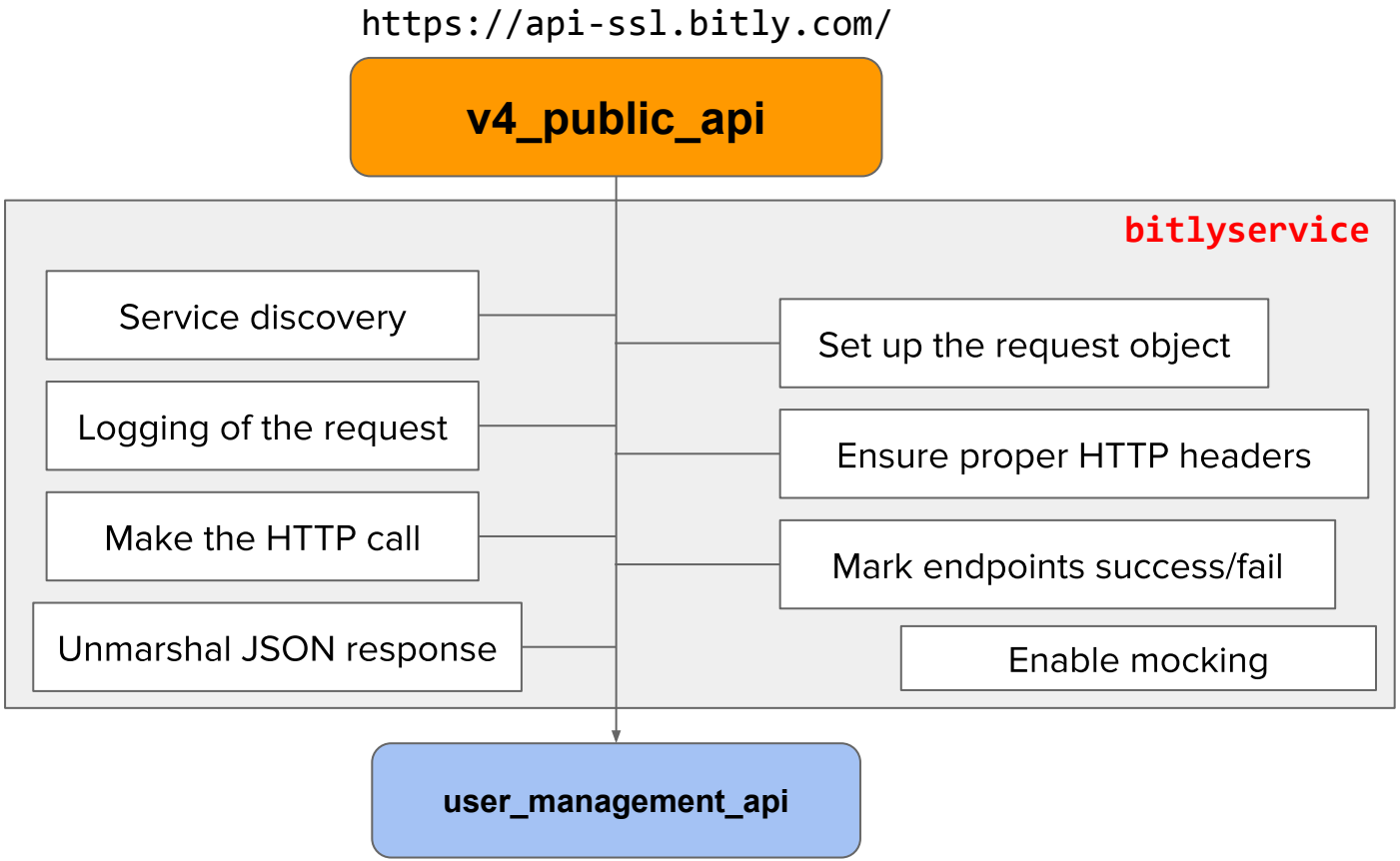
Test/mock support

Solution - Encapsulate it all into a reusable package

Enter `bitlyservice`

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
func (c client) GetUser(ctx context.Context, userGUID string) *usermanagementapi.User {
    req, err := bitlyservice.NewRequest(ctx, "/users/%s", nil, nil, userGUID)
    var user *usermanagementapi.User
    err = c.bitlyAPIs.Get("user_management_api", req, &user)
    return user
}
```

Under the covers - `bitlyservice` does all the work



Let me see the code

The Code

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
```

First, **`bitlyservice.NewRequest()`**

`bitlyservice.NewRequest()` – What does it do?

```
req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
```

In English: "Give me a **request object** that will call the endpoint "/orgs/orgGUID" with **no request body** and **no query params**."

```
func NewRequest(ctx context.Context, urlFormat string, body io.Reader, q *url.Values, pathParams
...string) (*http.Request, error) {

        To the terminal!

}
```

Returns an `http.Request` object

Next, let's unpack `Get()`

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
```

First piece: Service Discovery

```
var org *usermanagementapi.Org
err = c.bitlyAPIs.Get("user_management_api", req, &org)
```

`"user_management_api"`

What is that?? Where are the **host** and **port**??

named API: map a string like "user_management_api" to a set of hosts and ports

```
func (ba *BitlyAPI) Get(apiName string, r *http.Request, v interface{}) error {}
```

Application Startup

```
err = c.bitlyAPIs.Get("user_management_api", req, &org)
```

The above code is made possible by a registering of the **named API** at app start as seen below.

```
package main

func main() {
    // ...
    c.bitlyAPIs := bitlyservice.NewBitlyAPI("v4_api", nil)

    // Add a NamedAPI for each of the services you will need
    c.bitlyAPIs.MustAddNamedAPI("user_management_api")
}
```

This line ensures that our settings file has the correct host pool and port info for Named API "user_management_api".

Service will panic on start if settings are missing.

settings.json

```
"user_management_api": {"prefix": "http://", "role": "user_management_api.gc4", "port": 6931}
```

Service Validation - `MustAddNamedAPI()`

What if you pass an invalid api name?

```
c.bitlyAPIs.MustAddNamedAPI("user_management_api")
```

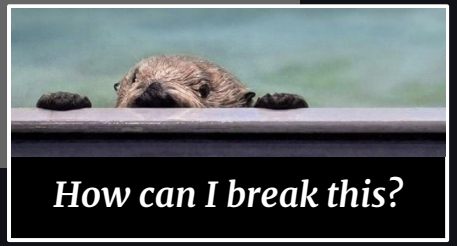
```
func (ba *BitlyAPI) MustAddNamedAPI(name string, hosts ...string) {

        To the terminal!

}
```

Service Discovery, check!

I know you want to see the `Get()` call, but first a word about

Supporting Testing/Mocking

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
```



Unit testing with hooks

```
func (t *testing.T) TestGetOrg() {
    mockGetOrg:= func(w http.ResponseWriter, r *http.Request) {
        apiresponse.OK200(w, testOrgInstance)
    }

    bAPI := bitlyservice.NewBitlyAPI("test", nil)
    bAPI.MustAddNamedAPI("user_management_api")
    bAPI.MustSetHook("user_management_api", mockGetOrg)

    bAPI.GetOrg("testGUID")  // this will return testOrgInstance
}
```

You didn't forget testing? Hallelujah!

As we shall see, this will cause the actual HTTP call to be circumvented

Digging into the actual HTTP call

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
```

The HTTP call and all things surrounding it

Now to the guts of the actual http request

```
var org *usermanagementapi.Org
err = c.bitlyAPIs.Get("user_management_api", req, &org)
```

Convenience functions exist for `Get()`, `Put()`, `Post()`, `Patch()` and `Delete()`.

```
func (ba *BitlyAPI) Get(apiName string, r *http.Request, v interface{}) error {

    // ba.Get() calls ba.do() which calls NamedAPI.Do() which calls NamedAPI.do().
    // Let's look...

        To the terminal!

}
```

Climbing back out...

What if we had to do that in every location in our code where we had copied stdlib `http` implementation?

```
func GetOrg(orgGUID string) *usermanagementapi.Org {
    client := &http.Client{}
    req, err := http.NewRequest("GET", "https://usermanagementapi01.bitly.com/orgs/michae
    req.Header.Add("X-Bitly-User-Agent", "v4_public_api")
    resp, err := client.Do(req)
    b, err := ioutil.ReadAll(resp.Body)
    var org *usermanagementapi.Org
    err = jsonsafe.Unmarshal(b, &org)
    return org
}
```

No, that would not be fun

Before & After (plus all the bells & whistles under the hood)

```
func GetOrg(orgGUID string) *usermanagementapi.Org {
    client := &http.Client{}
    req, err := http.NewRequest("GET", "https://usermanagementapi01.bitly.com/orgs/michaels-org", nil)
    req.Header.Add("X-Bitly-User-Agent", "v4_public_api")
    resp, err := client.Do(req)
    b, err := ioutil.ReadAll(resp.Body)
    var org *usermanagementapi.Org
    err = jsonsafe.Unmarshal(b, &org)
    return org
}
```

```
func (c client) GetOrg(ctx context.Context, orgGUID string) *usermanagementapi.Org {
    req, err := bitlyservice.NewRequest(ctx, "/orgs/%s", nil, nil, orgGUID)
    var org *usermanagementapi.Org
    err = c.bitlyAPIs.Get("user_management_api", req, &org)
    return org
}
```

Write Once, Use Many...

This is the power of encapsulation

```
dev ~/b:(bitly_master)$ ack --type go bitlyservice.NewRequest | wc -l
336
dev ~/b:(bitly_master)$
```

```
dev ~/b/v4_api:(bitly_master)$ ack 'GetOrg\('
internal/usermanagementapi/testclient/organizations.go
82:func (c Client) GetOrg(context.Context, string) (*models.Organization, error) {

internal/usermanagementapi/usermanagementapi.go
92:     GetOrg(context.Context, string) (*models.Organization, error)

v4_api/handlers/billing/billing.go
217:    org, err := h.um.GetOrg(ctx, orgGUID)

v4_api/handlers/organizations/organizations.go
667:            org, err := a.um.GetOrg(ctx, billingReq.OrgGUID)
1109:           org, err := a.um.GetOrg(ctx, nofitfyReq.OrgGUID)

v4_public_api/handlers/users/users.go
845:            org, err := h.um.GetOrg(ctx, invite.InvitedOrg)

v4_public_api/handlers/billing/billing.go
537:    org, err := h.um.GetOrg(ctx, orgGUID)
dev ~/b/v4_api:(bitly_master)$
```

I like this

And that's it...

- **Best practice: Encapsulation — write it once, use it everywhere**
- Clean, concise, consistent
- One place to control all intra-service API calls
- Changes roll out everywhere
- Avoids copy/paste/tweak proliferation of code

Thanks.

Lots of people at Bitly have contributed to this package.
The first pass was written by Kyle Purdon and Jehiah Czebotar.

@mrwoofster

https://bitly.is/hiring