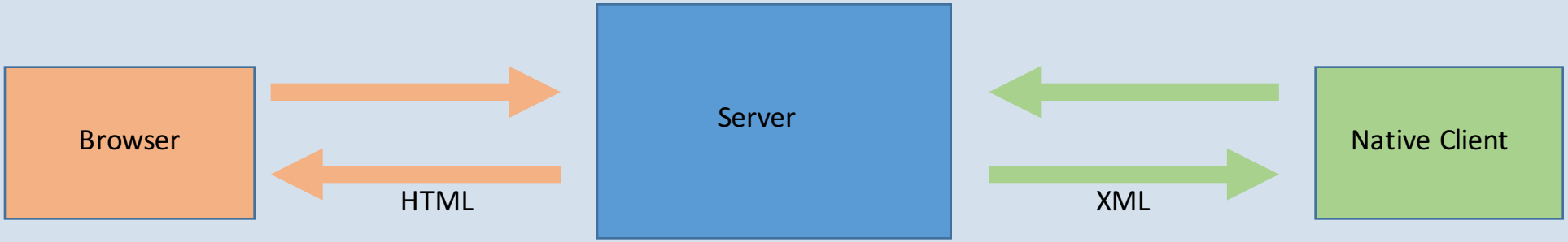
Symmetric API Testing

Aditya Mukerjee
@ChimeraCoder

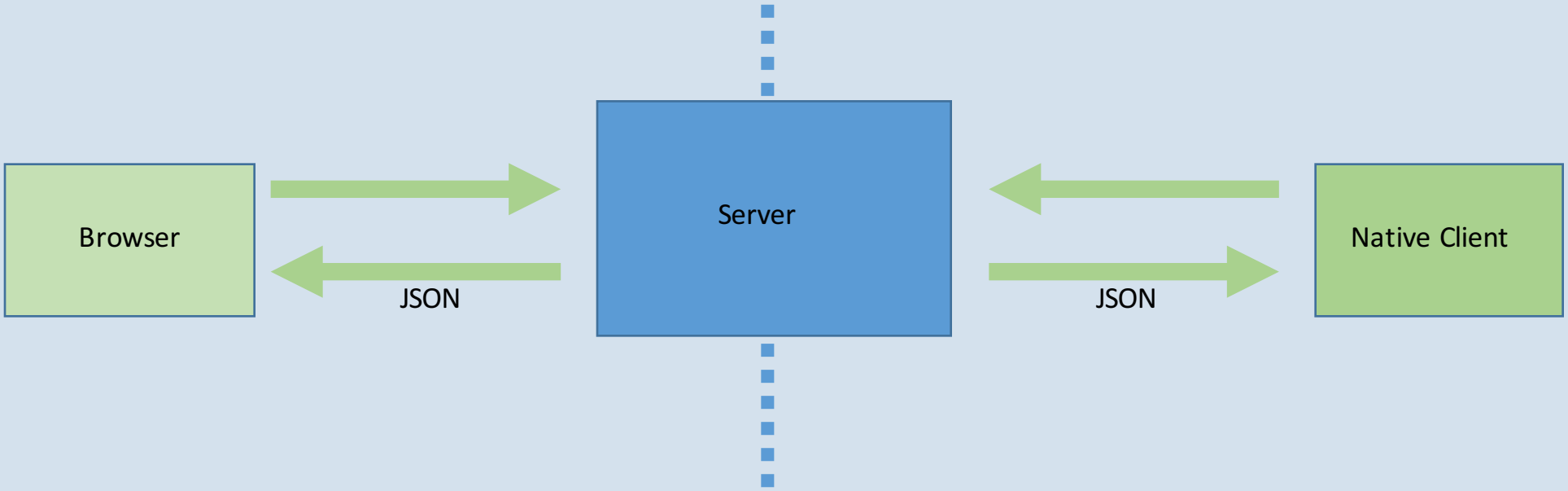
GORUCO 2016


stripe

So you want to build a webapp (2006)



So you want to build a webapp (2016)



Building a Webapp

- Design your API endpoints first
- Storyboard
- Develop your client and server together

Traditional API Testing

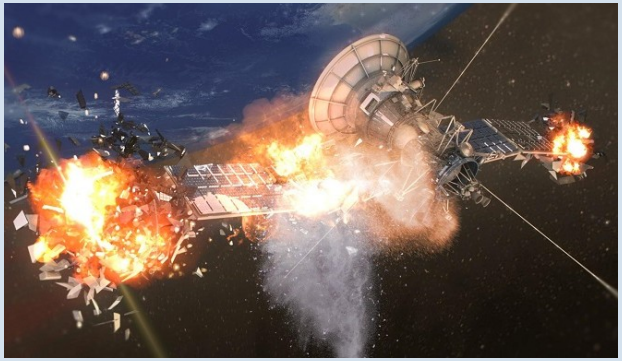
- Unit Tests
- Integration Tests
- Functional Tests

Advantages of Integration Tests

- Represent the behavior your users will actually experience
- Test the way your code **really** works, not the way you think the pieces **should** work
- Alert you to breaking changes that occur in the external environment
  - Depends on the test

Disadvantages of Integration Tests

- Fragile
- Reliant on external systems
- Can fail due to reasons outside your project's scope or responsibility



Do we have to choose?

- Most projects write both unit tests and integration/functional tests
- Duplication provides better coverage, but more code to maintain

README.md

Anaconda

Anaconda is a simple, transparent Go package for accessing version 1.1 of the Twitter API.

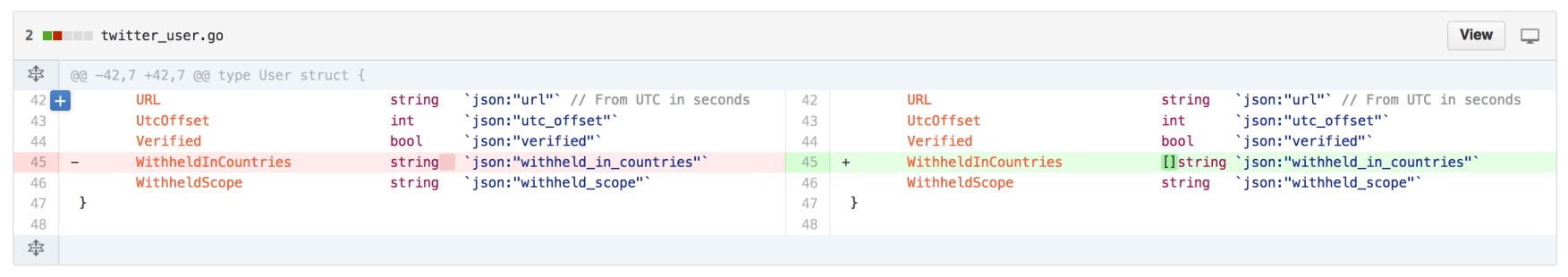
Successful API queries return native Go structs that can be used immediately, with no need for type assertions.

Mocking Responses

```
{
 "contributors_enabled": false,
 "created_at": "Sat Dec 14 04:35:55 +0000 2013",
 "default_profile": false,
 "default_profile_image": false,
```

```
type User struct {
        ContributorsEnabled             bool
        CreatedAt                       string
        DefaultProfile                  bool
        DefaultProfileImage             bool
        Description                     string
        Entities                        Entities
```

Perils of Stubbing and Mocking



2 twitter_user.go — View

```
@@ -42,7 +42,7 @@ type User struct {
```

42	+	URL	string	`json:"url"` // From UTC in seconds	42		URL	string	`json:"url"` // From UTC in seconds
43		UtcOffset	int	`json:"utc_offset"`	43		UtcOffset	int	`json:"utc_offset"`
44		Verified	bool	`json:"verified"`	44		Verified	bool	`json:"verified"`
45	-	WithheldInCountries	string	`json:"withheld_in_countries"`	45	+	WithheldInCountries	[]string	`json:"withheld_in_countries"`
46		WithheldScope	string	`json:"withheld_scope"`	46		WithheldScope	string	`json:"withheld_scope"`
47		}			47		}		
48					48				

What if our unit tests and our integration tests used the same code?

Symmetric API Testing (Client-Side)

Serving responses with a local server

```
mux := http.NewServeMux()
server := httptest.NewServer(mux)
parsed, _ := url.Parse(server.URL)

api.SetBaseUrl(parsed.String() + "/")

mux.HandleFunc("/myendpoint", func(w http.ResponseWriter, r *http.Request){
 fmt.Fprint(w, `<insert sample response here>`)
})
```

Eavesdropping on the live responses

```
resp, _ := oauthClient.Get(c.HttpClient, c.Credentials, urlStr, form)
```

```
return json.NewDecoder(resp.Body).Decode(data)
```

Eavesdropping on the live responses

```
resp, _ := oauthClient.Get(c.HttpClient, c.Credentials, urlStr, form)

resp.Body = io.TeeReader(resp.Body, file)

return json.NewDecoder(resp.Body).Decode(data)
```

Eavesdropping on the live responses

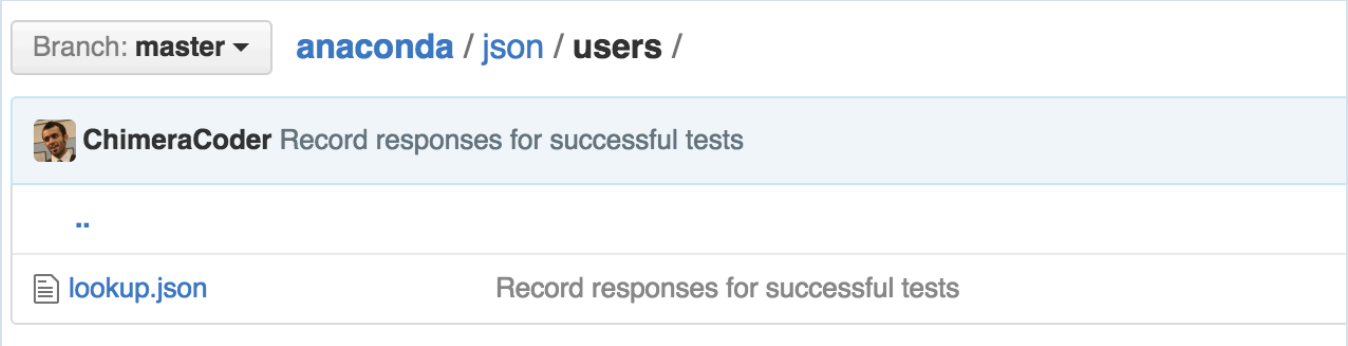
```
resp, _ := oauthClient.Get(c.HttpClient, c.Credentials, urlStr, form)

if !livemode {
        resp.Body = ioutil.NopCloser(io.TeeReader(resp.Body, f))
}

return json.NewDecoder(resp.Body).Decode(data)
```

Naming files

- REST-ful HTTP APIs give us a natural way to organize our recorded responses in our project's directory hierarchy



Generating structs with tooling

- Go structs are isomorphic to valid JSON objects
- Why do bookkeeping manually when computers can do it for us?

gojson

- gojson generates struct definitions from JSON documents

```
$ curl -s https://api.github.com/repos/chimeracoder/gojson | gojson -name=Repository
```

```
package main
type Repository struct {
        ArchiveURL string `json:"archive_url"`
        AssigneesURL string `json:"assignees_url"`
        ...
```

```
//go:generate gojson -o tweet.go -name "Tweet" -pkg "anaconda" –input
        json/statuses/show.json
```

- Insert this line in any file
- Running `go generate` will update all definitions across the project

Interfaces in Go (and elsewhere)

Interface Types in Go

```
type Writer interface {
   Write(p []byte) (n int, err error)
}

type Reader interface {
   Read(p []byte) (n int, err error)
}
```

```
type Foo struct {
      Body string
}


func (f Foo) Read(p []byte) (n int, err error){
      return copy(p, []byte(f.Body)), nil
}
```

```
func parseFile(input io.Reader) (Config, error){
    // who knows what this does to the file?
}
```

Interfaces are **powerful**

Interfaces provide **symmetry**

Symmetric API Testing (Server-Side)

Server-Side Testing

- Record the **actual** client requests
- Feed the recordings directly to the handlers on the server.

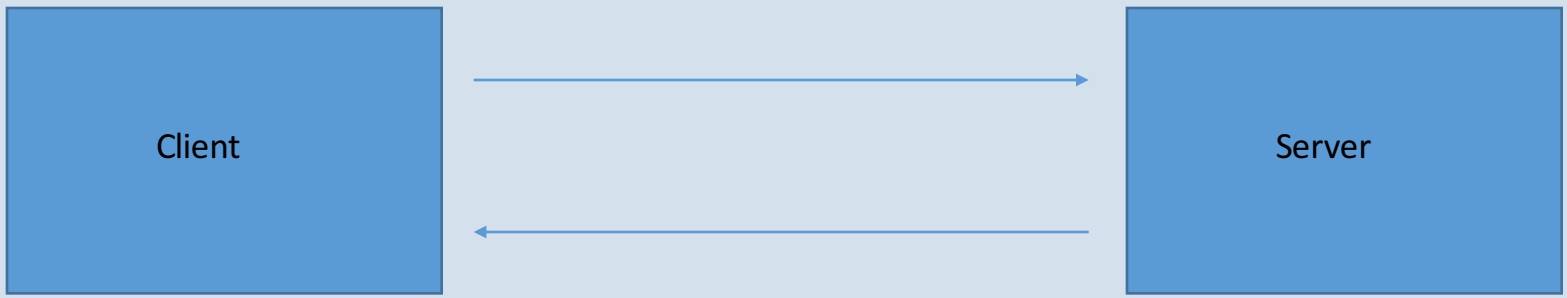
```
// this is the handler (controller) we want to test
func myHandler(w http.ResponseWriter, r *http.Request) {
      fmt.Fprint(w, "asdf")
}

// our test code
w := httptest.NewRecorder()
r, _ := http.NewRequest("POST", "http://example.com", data)

myHandler(w, r)
```

- Test for appropriate state changes (e.g. database state)
- Validate the responses returned by the server
  - But wait, haven't we already done something with the server responses?

- Client **and** server use each others' recordings in their tests
- Client **and** server use the **interface contract** for type definitions



What about other tools?

- Protobuf
- XML
- Thrift

Tenets of Symmetric API Testing

- APIs are composable and degrade gracefully. Tests should too.
- When designing an API, create at least one client for it.
- Lowering the barrier to writing tests increases coverage and reduces decay
- All endpoints should have a baseline testing, on both client and server.

Interfaces are **powerful**

Interfaces are the **axis of symmetry**

Aditya Mukerjee

@chimeracoder

github.com/ChimeraCoder