JSON REST API & GO

Opinionated talk about easy and rapid API development with Go

delivering web and mobile applications with the right attitude

HEAD WARNING

Following slides contain code!

DEV ENV

- Docker
- Makefile & bash
- Sublime Text 3

DOCKER

```
db:
  image: docker.io/postgres:9.5
  volumes_from:
    - dbData
  hostname: db
  ports:
    - 0.0.0.0:5432:5432

dbData:
  image: debian
  volumes:
    - /var/lib/postgresql/data
```

+ docker debian image if prefer your golang dev env containerized

MAKEFILE / I

```
GOROOT ?= /usr/local/go
GOPATH  = ${PWD}
GOBIN  ?= ${GOPATH}/bin

API_CFG  ?= ${PWD}/api.yml

export GOPATH
export API_CFG

build.api: deps
    go build -o bin/api github.com/kendu/meetupapp/commands/gateway

watch.api:
    ${GOBIN}/rerun github.com/kendu/meetupapp/commands/gateway

clean:
    docker-compose stop
    docker-compose rm -f
    rm -rf bin src/github.com src/golang.org src/gopkg.in pkg build
```

MAKEFILE /2

```
deps:
    grep -v "^#" dep.pkg|while read DEP; do echo $$DEP; go get $$DEP; done

goconvey:
     # Open browser @ http://127.0.0.1:8080/
    ${GOBIN}/goconvey -workDir src/git.kendu.si/kendu/meetupsample

test:
    go test `cd src; find git.kendu.si -type f -name '*_test.go'|xargs -n1 dirname|uniq`

fmt:
    find src/github.com/kendu/meetupapp -name '*.go' | xargs -n1 go fmt

vet:
    find src/github.com/kendu/meetupapp -name '*.go' | xargs -n1 go vet

qa: fmt vet test
```

DEPENDENCIES

gin-gonic/**gin**

RangelReale/**osin**

skelterjohn/**rerun**

dgrijalva/**jwt-go**

smartystreets/**goconvey**

gopkg.in/**mgutz/dat.v1**

xeipuuv/**gojsonschema**

mattes/**migrate**

RAML

"The simplest way to design APIs"

RAML + example files + json schema files = testing automation heaven

e2e testing stubs

validated examples

request validation

response validation

GIN WEB FRAMEWORK

- simple & fast
- easy to use/add/extend middleware
- route handling & grouping
- error management
- takes care of all my problems (like the 3rd round of gin)

ROUTING & MIDDLEWARE

```
func InitContactsController(di *registry.Registry) *ContactsController {
    ctrl := &ContactsController{}

    c := router.Group("/contacts", oauth2checker)
    {
        c.GET("", ctrl.Index)
        c.GET("/:contactId", ctrl.preload, ctrl.Read)
        c.DELETE("/:contactId", ctrl.preload, ctrl.Delete)
        c.POST("", checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Create)
        c.PUT("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
        c.PATCH("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
    }

    return ctrl
}
```

ROUTING & MIDDLEWARE

```
func InitContactsController(di *registry.Registry) *ContactsController {
    ctrl := &ContactsController{}

    c := router.Group("/contacts", oauth2checker)
    {
        c.GET("", ctrl.Index)
        c.GET("/:contactId", ctrl.preload, ctrl.Read)
        c.DELETE("/:contactId", ctrl.preload, ctrl.Delete)
        c.POST("", checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Create)
        c.PUT("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
        c.PATCH("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
    }

    return ctrl
}
```

ROUTING & MIDDLEWARE

```
func InitContactsController(di *registry.Registry) *ContactsController {
    ctrl := &ContactsController{}

    c := router.Group("/contacts", oauth2checker)
    {
        c.GET("", ctrl.Index)
        c.GET("/:contactId", ctrl.preload, ctrl.Read)
        c.DELETE("/:contactId", ctrl.preload, ctrl.Delete)
        c.POST("", checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Create)
        c.PUT("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
        c.PATCH("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
    }

    return ctrl
}
```

ROUTING & MIDDLEWARE

```
func InitContactsController(di *registry.Registry) *ContactsController {
    ctrl := &ContactsController{}

    c := router.Group("/contacts", oauth2checker)
    {
        c.GET("", ctrl.Index)
        c.GET("/:contactId", ctrl.preload, ctrl.Read)
        c.DELETE("/:contactId", ctrl.preload, ctrl.Delete)
        c.POST("", checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Create)
        c.PUT("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
        c.PATCH("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
    }

    return ctrl
}
```

Static format check using JSON schema

ROUTING & MIDDLEWARE

```
func InitContactsController(di *registry.Registry) *ContactsController {
    ctrl := &ContactsController{}

    c := router.Group("/contacts", oauth2checker)
    {
        c.GET("", ctrl.Index)
        c.GET("/:contactId", ctrl.preload, ctrl.Read)
        c.DELETE("/:contactId", ctrl.preload, ctrl.Delete)
        c.POST("", checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Create)
        c.PUT("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
        c.PATCH("/:contactId", ctrl.preload, checkJson("contacts.item.request.POST"), ctrl.parseInput, ctrl.Update)
    }

    return ctrl
}
```

Load input into internal values + preform dynamic validation

MIDDLEWARE

```
func (ctrl *ContactsController) preload(c *gin.Context) {
    contact, err := ctrl.contactService().FindById(c.Param("contactId"))

    if err != nil || contact.Deleted.Valid {
        c.AbortWithError(http.StatusNotFound, ContactNotFound)
    } else {
        c.Set("preloaded", contact)
    }
}
```

MIDDLEWARE

```
func (ctrl *ContactsController) parseInput(c *gin.Context) {
    contact := &models.Contact{}

    if c.Request.Method == "PATCH" {
        // When patching, use preloaded data to merge
        contact = c.MustGet("preloaded").(*models.Contact)
    }

    input := struct {/* temp input struct */}{}

    err := c.BindJSON(&input)

    if err != nil {
        c.Error(err)
        c.AbortWithError(http.StatusBadRequest, ContactInvalidInputError)
        return
    }

     // mapping parsed input to internal structure

    c.Set("parsed", contact)
}
```

THIN(NER) CONTROLLERS

```
func (ctrl *ContactsController) Create(c *gin.Context) {
    fresh := c.MustGet("parsed").(*models.Contact)

    if err := ctrl.contactService().Create(fresh); err != nil {
        c.Error(err)
        c.AbortWithError(http.StatusInternalServerError, ContactsInternalError)
    } else {
        c.JSON(http.StatusOK, fresh)
    }
}


func (ctrl *ContactsController) Read(c *gin.Context) {
    c.JSON(http.StatusOK, c.MustGet("preloaded"))
}
```

TESTING WITH GO COVEY

Works perfectly with `go test` + has **auto-updating web UI** for test results

```
func TestCsvFetching(t *testing.T) {
    Convey("Contacts can be downloaded as CSV", t, func() {
        client := &http.Client{}

        req, err := http.NewRequest("GET", serverUrl()+"/contacts", nil)
        req.Header.Add("accept", "text/csv")
        rsp, err := client.Do(req)
        So(err, ShouldBeNil)

        csv := ""
        handleResponse(rsp, &csv, vs(http.StatusOK))

        So(csv, ShouldStartWith, "to,id")
        So(csv, ShouldContainSubstring, contact1.Email)
        So(csv, ShouldContainSubstring, contact1.Id)
    })
}
```

TESTING CONTROLLERS

```
func TestMain(m *testing.M) {
    server = httptest.NewServer(router)
    defer server.Close()

    os.Exit(m.Run())
}
```

TESTING CONTROLLERS

```
func get(endpoint string, rval interface{}, expectedStatuses []int) {
    rsp, err := http.Get(endpoint)
    So(err, ShouldBeNil)
    handleResponse(rsp, rval, expectedStatuses)
}

func handleResponse(rsp *http.Response, rval interface{}, expectedStatuses []int) {
    if rsp.StatusCode != http.StatusNoContent {
        body, err = ioutil.ReadAll(rsp.Body)
        rsp.Body.Close()
        So(err, ShouldBeNil)
    }

    So(rsp.StatusCode, ShouldBeIn, expectedStatuses)

    if rsp.StatusCode == http.StatusNoContent { return }

    // ignoring edge cases…

    So(body, shouldBeValidJson)
    err = json.Unmarshal(body, rval)
    So(err, ShouldBeNil)
}
```

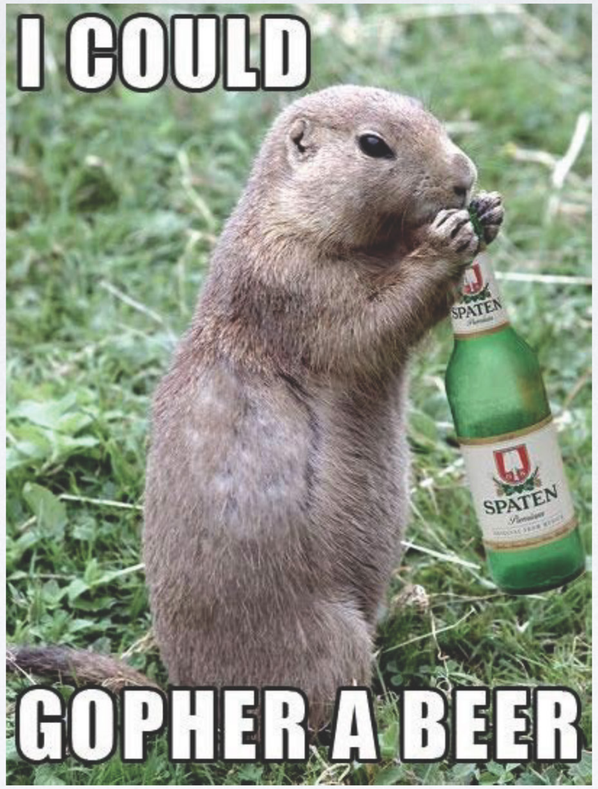
TESTING CONTROLLERS

```
Convey("A contact can be modified", func() {
    do("PUT", url, `{"email":"test-modified@foo.net","data":{"key":"val"}}`, &contact, vs(http.StatusOK))
    get(url, &contact, vs(http.StatusOK))
    So(contact.Email, ShouldEqual, “test-modified@foo.net")
    So(contact.Data, ShouldContainKey, "key")
    So(contact.Data["key"], ShouldEqual, "val")
})

Convey("A contact can be patched", func() {
    do("PATCH", url, `{“email":"test-modified-2@foo.net"}`, &contact, vs(http.StatusOK))
    get(url, &contact, vs(http.StatusOK))
    So(contact.Email, ShouldEqual, “test-modified-2@foo.net")
    So(contact.Data, ShouldContainKey, "key")
})
```

LESSONS LEARNED

- don’t panic()
- implicit interfaces are awesome
- don’t get bitten by pointers and references



placed here for sole purpose of stealing thunder of the last speaker

QUESTIONS?