

3 - Making Your API RESTful

Outlining the API

- Resource - a list of users
- List item data - name, role, and id
- Functionality - Create, Read, Update, and Delete (CRUD)
- Endpoints:
 - Collection: /users
 - Item: /users/{id}
- Data format - JSON

Create an Item

- Method - POST
- Target - collection
- Endpoint - POST/users
- Request content - full item data
- Successful response - 201 created + location

Access the Collection

- Method - GET
- Target - collection
- Endpoint - GET/users
- Request content - none
- Successful response - 200 OK + list of users
- Missing resource - 404 not found

Replace an Item

- Method - PUT
- Target - item
- Endpoint - PUT /user/{id}
- Request content - full item data
- Successful response - 200 OK + new item data
- Missing resource - 404 not found

Update an Item

- Method - PATCH
- Target - item
- Endpoint - PATCH /users/{id}
- Request content - partial item data
- Successful response - 200 OK + new item data
- Missing resource - 404 not found

Typical Responses

- Malformed requests - 400 bad request
- Execution errors - 500 internal server error
- Invalid method - 405 method not allowed
- Missing resources - 404 not found

Setting Up a Data Store

Persistent Data Store

- Local key-value store:
 - Low traffic volume, little data, and simple data
- Separate DB server:
 - Complex data, independently scalable, and busier sites
- Memory cache + DB cluster
 - High-end, big data, and high traffic volume

Option 1: storm

```
1 $ go get github.com/asdine/storm
```

Option 2; bson

```
1 $ go get gopkg.in/mgo.v2/bson
```

src/user/main.go

```
1 package user
2
3 import "gopkg.in/mgo.v2/bson"
4
5 // User holds data for a single user
6 type User struct {
7     ID    bson.ObjectId `json:"id" storm:"id"`
8     Name string        `json:"name"`
9     Role string        `json:"role"`
10 }
```

Record Manipulation

src/user/main.go

```
1 package user
2
3 import (
4     "errors"
5
6     "github.com/asdine/storm"
7     "gopkg.in/mgo.v2/bson"
8 )
9
10 // User holds data for a single user
11 type User struct {
12     ID      bson.ObjectId `json:"id" storm:"id"`
13     Name    string        `json:"name"`
14     Role    string        `json:"role"`
15 }
16
17 const (
18     dbPath = "user.db"
19 )
20
21 // errors
22 var (
23     ErrRecordInvalid = errors.New("record is invalid")
24 )
25
26 // All retrieves all users from the database
27 func All() ([]User, error) {
28     db, err := storm.Open(dbPath)
29     if err != nil {
30         return nil, err
31     }
32     defer db.Close()
33     users := []User{}
34     err = db.All(&users)
35     if err != nil {
36         return nil, err
37     }
38     return users, nil
39 }
40
41 // One returns a single user record from the database
42 func One(id bson.ObjectId) (*User, error) {
43     db, err := storm.Open(dbPath)
44     if err != nil {
45         return nil, err
46     }
47     defer db.Close()
48     u := new(User)
49     err = db.One("ID", id, u)
50     if err != nil {
51         return nil, err
52     }
53     return u, nil
54 }
```

```

55
56 // Delete removes a given record from the database
57 func Delete(id bson.ObjectId) error {
58     db, err := storm.Open(dbPath)
59     if err != nil {
60         return err
61     }
62     defer db.Close()
63     u := new(User)
64     err = db.One("ID", id, u)
65     if err != nil {
66         return err
67     }
68     return db.DeleteStruct(u)
69 }
70
71 // Save updates or creates a given record in the database
72 func (u *User) Save() error {
73     if err := u.validate(); err != nil {
74         return err
75     }
76     db, err := storm.Open(dbPath)
77     if err != nil {
78         return err
79     }
80     defer db.Close()
81     return db.Save(u)
82 }
83
84 // validate makes sure that the record contains valid data
85 func (u *User) validate() error {
86     if u.Name == "" {
87         return ErrRecordInvalid
88     }
89     return nil
90 }

```

Creating a Custom Handler

API Resource Requests

- Common route - /users
- Algorithm:
 - Recognize the /users route
 - Parse the URL and method to call a proper handler
 - Retrieve or process data
 - Send the response

src/handlers/rootHandler.go

```

1 package handlers
2
3 import "net/http"

```

```

4
5 // RootHandler handles the root route
6 func RootHandler(w http.ResponseWriter, r *http.Request) {
7     if r.URL.Path != "/" {
8         w.WriteHeader(http.StatusNotFound)
9         w.Write([]byte("Asset not found\n"))
10        return
11    }
12    w.WriteHeader(http.StatusOK)
13    w.Write([]byte("Running API v1\n"))
14 }

```

src/handlers/usersRouter.go

```

1 package handlers
2
3 import (
4     "net/http"
5     "strings"
6
7     "gopkg.in/mgo.v2/bson"
8 )
9
10 // UsersRouter handles the users route
11 func UsersRouter(w http.ResponseWriter, r *http.Request) {
12     path := strings.TrimSuffix(r.URL.Path, "/")
13
14     if path == "/users" {
15         switch r.Method {
16             case http.MethodGet:
17                 return
18             case http.MethodPost:
19                 return
20             default:
21                 postError(w, http.StatusMethodNotAllowed)
22                 return
23         }
24     }
25
26     path = strings.TrimPrefix(path, "/users/")
27     if !bson.IsObjectIdHex(path) {
28         postError(w, http.StatusNotFound)
29         return
30     }
31
32     // id := bson.ObjectIdHex(path)
33
34     switch r.Method {
35         case http.MethodGet:
36             return
37         case http.MethodPut:
38             return
39         case http.MethodPatch:
40             return
41         case http.MethodDelete:
42             return
43         default:

```

```

44     postError(w, http.StatusMethodNotAllowed)
45     return
46 }
47 }
48

```

src/main.go

```

1 package main
2
3 import (
4     "fmt"
5     "net/http"
6     "os"
7
8     "github.com/build-restful-apis/src/handlers"
9 )
10
11 func main() {
12     http.HandleFunc("/users/", handlers.UsersRouter)
13     http.HandleFunc("/users", handlers.UsersRouter)
14     http.HandleFunc("/", handlers.RootHandler)
15     err := http.ListenAndServe("localhost:8080", nil)
16     if err != nil {
17         fmt.Println(err)
18         os.Exit(1)
19     }
20 }

```

src/handlers/responses.go

```

1 package handlers
2
3 import "net/http"
4
5 func postError(w http.ResponseWriter, code int) {
6     http.Error(w, http.StatusText(code), code)
7 }

```

testcase 1:

```
1 $ curl localhost:8080
```

testcase 2:

```
1 $ curl localhost:8080/users
```

testcase 3:

```
1 $ curl localhost:8080/users/1
```

Retrieving a List (GET)

GET Request

- Responses:

- 200 OK + a list of items in the collection
- 404 not found
- 500 internal server error

Response Content

A list or an unnamed item

Body:

```

1 [
2   {
3     "name": "Sandra",
4     "role": "team leader"
5   },
6   {
7     "name": "Joe",
8     "role": "developer"
9   }
10 ]

```

src/handlers/responses.go

```

1 package handlers
2
3 import (
4     "encoding/json"
5     "net/http"
6 )
7
8 type jsonResponse map[string]interface{}
9
10 func postError(w http.ResponseWriter, code int) {
11     http.Error(w, http.StatusText(code), code)
12 }
13
14 func postBodyResponse(w http.ResponseWriter, code int, content jsonResponse) {
15     if content != nil {
16         js, err := json.Marshal(content)
17         if err != nil {
18             postError(w, http.StatusInternalServerError)
19             return
20         }
21         w.Header().Set("Content-Type", "application/json")
22         w.WriteHeader(code)
23         w.Write(js)
24         return
25     }
26     w.WriteHeader(code)
27     w.Write([]byte(http.StatusText(code)))
28 }

```

src/handlers/useHandlers.go

```

1 func usersGetAll(w http.ResponseWriter, r *http.Request) {
2     users, err := user.All()

```

```


3     if err != nil {
4         postError(w, http.StatusInternalServerError)
5         return
6     }
7     postBodyResponse(w, http.StatusOK, jsonResponse{"users": users})
8 }

```

Creating an Item (POST)

POST Request

- Method - POST
- Target - a collection
- Request - full item content
- Result - an item is added to the collection

	State	Request	Result
POST		<pre>{ "name" : "Mike", "role" : "team leader" }</pre>	<pre>{ "name" : "Mike", "role" : "team leader" }</pre>
PUT	<pre>{ "name" : "Joe", "role" : "developer" }</pre>	<pre>{ "name" : "Sally" }</pre>	<pre>{ "name" : "Sally", "role" : "" }</pre>
PATCH	<pre>{ "name" : "Joe", "role" : "developer" }</pre>	<pre>{ "name" : "Sally" }</pre>	<pre>{ "name" : "Sally", "role" : "developer" }</pre>

Update a Record

- By assignment and duplication:
 - `recordItem, err = Update(data, recordItem) (*recordItem, error)`
- By reference:
 - `err = Update(data, *recordItem) error`

src/handlers/userHandlers.go

```

1 func usersPostOne(w http.ResponseWriter, r *http.Request) {
2     u := new(user.User)
3     err := bodyToUser(r, u)
4     if err != nil {
5         postError(w, http.StatusBadRequest)
6         return
7     }

```



```

8      u.ID = bson.NewObjectId()
9      err = u.Save()
10     if err != nil {
11         if err == user.ErrRecordInvalid {
12             postError(w, http.StatusBadRequest)
13         } else {
14             postError(w, http.StatusInternalServerError)
15         }
16         return
17     }
18     w.Header().Set("Location", "/users/"+u.ID.Hex())
19     w.WriteHeader(http.StatusCreated)
20 }

```

Retrieving an Item (GET)

src/handlers/userHandlers.go

```

1 func usersGetOne(w http.ResponseWriter, _ *http.Request, id bson.ObjectId) {
2     u, err := user.One(id)
3     if err != nil {
4         if err == storm.ErrNotFound {
5             postError(w, http.StatusNotFound)
6             return
7         }
8         postError(w, http.StatusInternalServerError)
9         return
10    }
11    postBodyResponse(w, http.StatusOK, jsonResponse{"user": u})
12 }

```

Replacing an Item (PUT)

PUT Request

- Method - PUT
- Target - an item
- Request - full item content
- Result - resource is replaced with the request body
- Responses:
 - 200 OK + updated resource
 - 400 bad request
 - 404 not found
 - 500 internal server error

src/handlers/userHandlers.go

```

1 func usersPutOne(w http.ResponseWriter, r *http.Request, id bson.ObjectId) {
2     u := new(user.User)
3     err := bodyToUser(r, u)
4     if err != nil {

```

```

5      postError(w, http.StatusBadRequest)
6      return
7  }
8  u.ID = id
9  err = u.Save()
10 if err != nil {
11     if err == user.ErrRecordInvalid {
12         postError(w, http.StatusBadRequest)
13     } else {
14         postError(w, http.StatusInternalServerError)
15     }
16     return
17 }
18 postBodyResponse(w, http.StatusOK, jsonResponse{"user": u})
19 }

```

Updating an Item (PATCH)

PATCH Request

- Method - PATCH
- Target - an item
- Request - partial item content
- Result - resource is updated based on fields existing in the request
- Responses:
 - 200 OK + updated resource
 - 400 bad request
 - 404 not found
 - 500 internal server error

src/handlers/userHandlers.go

```

1 func usersPatchOne(w http.ResponseWriter, r *http.Request, id bson.ObjectId) {
2     u, err := user.One(id)
3     if err != nil {
4         if err == storm.ErrNotFound {
5             postError(w, http.StatusNotFound)
6             return
7         }
8         postError(w, http.StatusInternalServerError)
9         return
10    }
11    err = bodyToUser(r, u)
12    if err != nil {
13        postError(w, http.StatusBadRequest)
14        return
15    }
16    u.ID = id
17    err = u.Save()
18    if err != nil {
19        if err == user.ErrRecordInvalid {
20            postError(w, http.StatusBadRequest)
21        } else {

```

```

22         postError(w, http.StatusInternalServerError)
23     }
24     return
25 }
26 w.Header().Set("Location", "/users/"+u.ID.Hex())
27 w.WriteHeader(http.StatusCreated)
28 }

```

Removing an Item (DELETE)

DELETE Request

- Method - DELETE
- Target - an item
- Request - no content
- Result - the resource is removed
- Responses:
 - 200 OK (sometimes 204 no content)
 - 404 not found
 - 500 internal server error

src/handlers/userHandlers/go

```

1 func usersDeleteOne(w http.ResponseWriter, _ *http.Request, id bson.ObjectId) {
2     err := user.Delete(id)
3     if err != nil {
4         if err == storm.ErrNotFound {
5             postError(w, http.StatusNotFound)
6             return
7         }
8         postError(w, http.StatusInternalServerError)
9         return
10    }
11    w.WriteHeader(http.StatusOK)
12 }

```

Retrieving Headers (HEAD)

HEAD Request

- Method - HEAD
- Target - any resource
- Request - no content
- Result - no effect on the resource
- Responses:
 - 200 OK + headers from GET method
 - 404 not found
 - 500 internal server error

src/handlers/userHandler.go

```
1 func usersGetOne(w http.ResponseWriter, r *http.Request, id bson.ObjectId) {
2     ...
3     if r.Method == http.MethodHead {
4         postBodyResponse(w, http.StatusOK, jsonResponse{})
5         return
6     }
7     postBodyResponse(w, http.StatusOK, jsonResponse{"user": u})
8 }
9
10 func usersGetAll(w http.ResponseWriter, r *http.Request) {
11     if r.Method == http.MethodHead {
12         postBodyResponse(w, http.StatusOK, jsonResponse{})
13         return
14     }
15     postBodyResponse(w, http.StatusOK, jsonResponse{"users": users})
16     ...
17 }
```

Retrieving Options (OPTIONS)

OPTIONS Request

- Method - OPTIONS
- Target - any resource
- Request - no content
- Result - no effect on the resource
- Responses:
 - 200 OK + “Allow” header with list of implemented methods
 - Optional response - body with documentation of allowed requests

src/handlers/responses.go

```
1 func postOptionsResponse(w http.ResponseWriter, methods []string, content jsonResponse) {
2     w.Header().Set("Allow", strings.Join(methods, ","))
3     postBodyResponse(w, http.StatusOK, content)
4 }
```

src/handlers/usersRouter.go

```
1 // UsersRouter handles the users route
2 func UsersRouter(w http.ResponseWriter, r *http.Request) {
3     ...
4     case http.MethodOptions:
5         postOptionsResponse(w, []string{ http.MethodGet, http.MethodPost,
6             http.MethodHead, http.MethodOptions }, nil)
7         return
8     default:
9         postError(w, http.StatusMethodNotAllowed)
10        return
11 }
```

```
12     ...
13
14     case http.MethodOptions:
15         postOptionsResponse(w, []string{ http.MethodGet, http.MethodPut, http.MethodPatch,
16             http.MethodDelete, http.MethodHead, http.MethodOptions }, nil)
17         return
18     default:
19         postError(w, http.StatusMethodNotAllowed)
20         return
21 }
22
```

Summary

- Outlined the API
- Created the data store
- Added CRUD functionality to a collection
- Created a custom router
- Added individual handlers
- Discussed documentation

