Software Engineering, EOSF322, Fall 2019
Creating and Testing Endpoints, Notes from Justin's Presentation

Each issue ought to map to a requirement.

Given an issue here are the steps to enable the web service to provide needed information for that requirement.

Example: AM6

AM6: View all semesters

Administrators shall be able to a view a list of all semesters in the system, ordered from the latest (most recent) semester to the earliest.

Rationale: Administrators need to know what semesters AbOut knows about.

Steps:

1. Create a view for the Semester table. Call that view SemesterView
   - This can be done directly with MySQL however in the long run, this needs to be added to the script AbOut_DB_createTables.sql. Use the sql command 'CREATE OR REPLACE VIEW viewName AS SELECT ….' Place the statements to create the view immediately after the table is created.

Within the file AbOut/backend/models/semesters.go:

2. A semester interface will contain definitions to all semester operations. Add a definition of the new GetSemester function to this interface.
   - The interface is located in backend/models/semesters.go
   - Add GetSemesters() (Semesters, error) ~line 12
     - No inputs
     - Returns a collection of semester objects and an error

3. If not already defined, need a Semester struct for the semester objects returned.

```
// Semester a model for the Semesters table in the db.  ~line 30-34
type Semester struct {
    SemesterID int
            `gorm:"type:int(11);Column:semesterId;AUTO_INCREMENT;not
                                    null;PRIMARY_KEY" json:"semesterId"`
    Semester   string
            `gorm:"type:char(12);not null;UNIQUE" json:"semester"`
}
```

   - Can look up correct syntax in GORM documentation. "Pluralize table name" syntax is helpful. The documentation says to return "SemesterView".

4. Write the GetSemesters() function so that it returns all of the semesters.

```
// GetSemesters will return all the semesters in the table     ~lines 45-53
func (SemesterRepo) GetSemesters() (Semesters, error) {
        Semester := []Semester{}
        err := db.Select("semesterId, semester").
                Order("semesterId desc").Find(&Semester).Error
                // Use the gorm documentation to know exactly
                // how to specify what we want.
                // Find(&Semesters) says to put the results at the
                // address of, &, semesters.
        if err != nil {
                return nil, err
        }
        return Semester, nil
}
```

- (SemesterRepo) indicates that GetSemesters() is a receiver function, that is, GetSemesters can receive an item from the SemesterRepo.
- In the body, declare the semester using notation for the struct (I think).
- If there is an error, just return it. Since this is a db error, don't compose anything for the user. Let the db error be shown.

Within the file AbOut/backend/server/router.go:

5. Tell the router to call GetSemesters() when users request this endpoint by adding the yellowed line to the servers router function.

```
// NewRouter creates a mux router and defines the routes for the application.
func NewRouter() *mux.Router {
        r := mux.NewRouter()
        sAPI := endpoints.SemesterAPI{Repo: models.SemesterRepo{}}
        r.HandleFunc("/semesters", sAPI.GetSemesters).    ~line 5
                Methods(http.MethodGet)
        r.HandleFunc("/semesters", sAPI.AddSemester).
                Methods(http.MethodPost)
        r.HandleFunc("/semesters/{id:[0-9]+}", sAPI.GetSemester).
                Methods(http.MethodGet)
        return r
}
```

Within the file AbOut/endpoints/semester.go:

6. Define the endpoint, which will be a GET

```
// GetSemesters returns all the semesters in the database.      ~lines 19-35
func (api SemesterAPI) GetSemesters(w http.ResponseWriter, r *http.Request) {
        semesters, err := api.Repo.GetSemesters()
        if err != nil {
                log.Println(err)
                w.WriteHeader(http.StatusInternalServerError)
                return
```

```
        }
        b, err := json.Marshal(semesters)
        if err != nil {
                log.Println(err)
                w.WriteHeader(http.StatusInternalServerError)
                return
        }
        OK(w)
        w.Write(b)
}
```

- All endpoints take a request, ‘w’ in the code above, and return a response, ‘r’.
- For this endpoint the request is empty. The code is building the response.

Within the file AbOut/endpoints/semester_test.go:

7. Set up a database mock.

```
func (mockRepo) GetSemesters() (models.Semesters, error) {   ~lines 26-38
        semesters := []models.Semester{
                {
                        SemesterID: 2,
                        Semester:   "Spring 2019",
                },
                {
                        SemesterID: 1,
                        Semester:   "Fall 2019",
                },
        }
        return semesters, nil
}
```

8. Create tests of the new endpoint. Here a test for the good case is created.

```
func TestGetSemesters_GoodCase_ResponseOK(t *testing.T) { ~lines 82-116
    // Arrange:
    // Create a request to send to the endpoint.
    req, err := http.NewRequest("GET", "/semesters", nil)
    if err != nil {
            t.Fatal(err)
    }

    // Create a response recorder to capture the response
    rr := httptest.NewRecorder()
    // Set the handler for the test
    handler := http.HandlerFunc(mockAPI.GetSemesters)
```

```
    // Act:
    // Perform the request.
    handler.ServeHTTP(rr, req)

    // Assert:
    // Check that the response code is as expected.
    if status := rr.Code; status != http.StatusOK {
            t.Errorf("handler returned wrong status code: got %v want %v",
                    status, http.StatusOK)
    }
    // Check that the response body is as expected.
    expected := `[{"semesterId":2,"semester":"Spring
2019"},{"semesterId":1,"semester":"Fall 2019"}]`
    if rr.Body.String() != expected {
            t.Errorf("handler returned unexpected body: got %v want %v",
                    rr.Body.String(), expected)
    }
    // Check that the headers are as expected.
    if ctype := rr.Header().Get(contentType); ctype != appjson {
            t.Errorf("content type header does not match: got %v want %v",
                    ctype, appjson)
    }
}
```

Hooray – ready to try!

This can be tested using Linux (lumen or Katie) or on your Windows machine. I'll test it on my Windows machine.

Install Go on your Windows machine if you haven't already.

1. Start the go server.

   CNRT-SHIFT ~ opens a Windows PowerShell within VS Code. One can be opened to run the server, and another to run the clients/tests.

   Start the go server via "go run main.go"

2. Create tests for http requests.
   Install two VS Code extensions.

REST Client	// Create and send http requests
MySQL	// Enable writing queries within the VS Code // environment. Icon

Within VS Code, at the same livel as main.go file, create the file test.http (this will not be uploaded to the GitLab repository because it is mentioned in GitLab's .ignore file). Within test.http create the GET semesters test:

```
GET http://localhost:8080/semesters HTTP/1.1
```

The REST Client extension will insert a "Send Request" link above this test. Click this link to run the test.

Separate tests with one or more blank lines, ### and one or more blank lines.

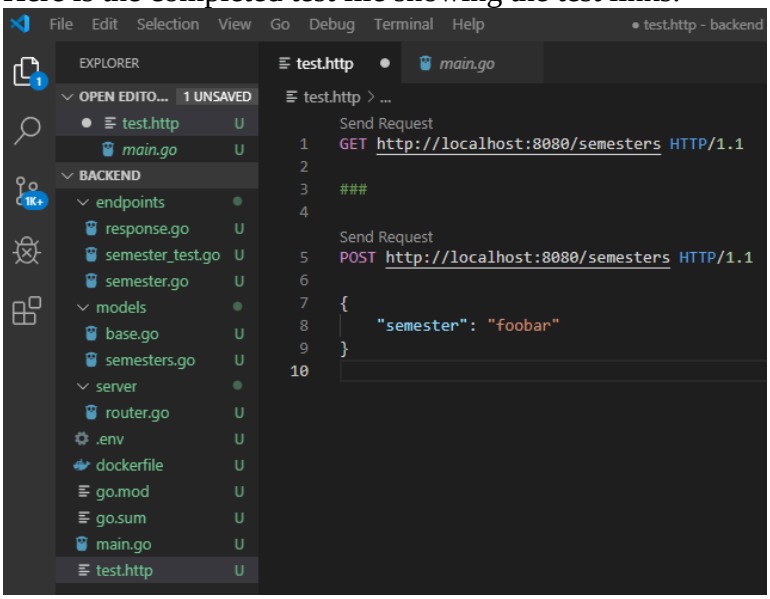
Sample POST test:

```
POST http://localhost:8080/semesters HTTP/1.1

{
    "semester": "Fall 2019"
}
```

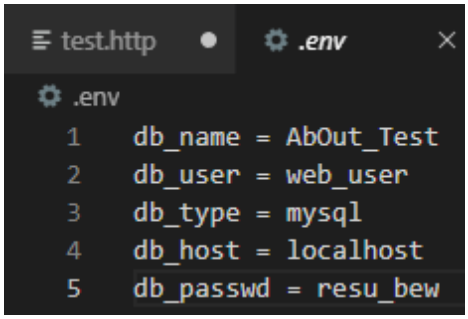
Note that the blank line after the POST is required. The format is very strict, as these must be exact HTTP requests.

Here is the completed test file showing the test links:



3. Click the line "Send Request" to test the endpoint.

The file server/.env file holds connection information:


test.http
.env
.env
1 db_name = AbOut_Test
2 db_user = web_user
3 db_type = mysql
4 db_host = localhost
5 db_passwd = resu_bew