Databases, JSON, and GORM

Software Engineering

Database

Database tables are not directly accessed.

- Create views
- db_docs/AbOut_DB_createTAbles.sql
- Example:

```
-- Addition of views for User
CREATE OR REPLACE VIEW ‘UserView’ AS
    SELECT userId, uName, usernameCAS, activeFlag
    FROM User;
```

DB Credentials

The server is not instantiated if the db credential are wrong.

>> go run main.go

```
Current\AbOut_Refactor\sprint2\AbOut\backend> go run main.go
        root:mysql@(mysql)/AbOut_Celia?charset=utf8    // The dbURI
                // username:password@(dbHost)dbName?charset=utf8
        dial tcp: lookup mysql: no such host       // Line 40 (see green)
        panic: sql: database is closed             // Line 46 (see blue)

        goroutine 1 [running]:
        gitlab.cs.mtech.edu/ESOF332/F19/AbOut/backend/models.init.0()
                C:/1Work/AB_Courses/ESOF_322/ESOF322---
        Current/AbOut_Refactor/sprint2/AbOut/backend/models/base.go:46 +0x562
        exit status 2
```

Server Not Instantiated Due to DB

The problem is occurs when AbOut/backend/models/base.go, line 46 calls:

```
// Init sets up the database connection.
func init() {
    err := godotenv.Load()
    if err != nil {
        err = godotenv.Load("../.env")
        if err != nil {
            fmt.Println(err)
        }
    }
```

Server Not Instantiated Due to DB

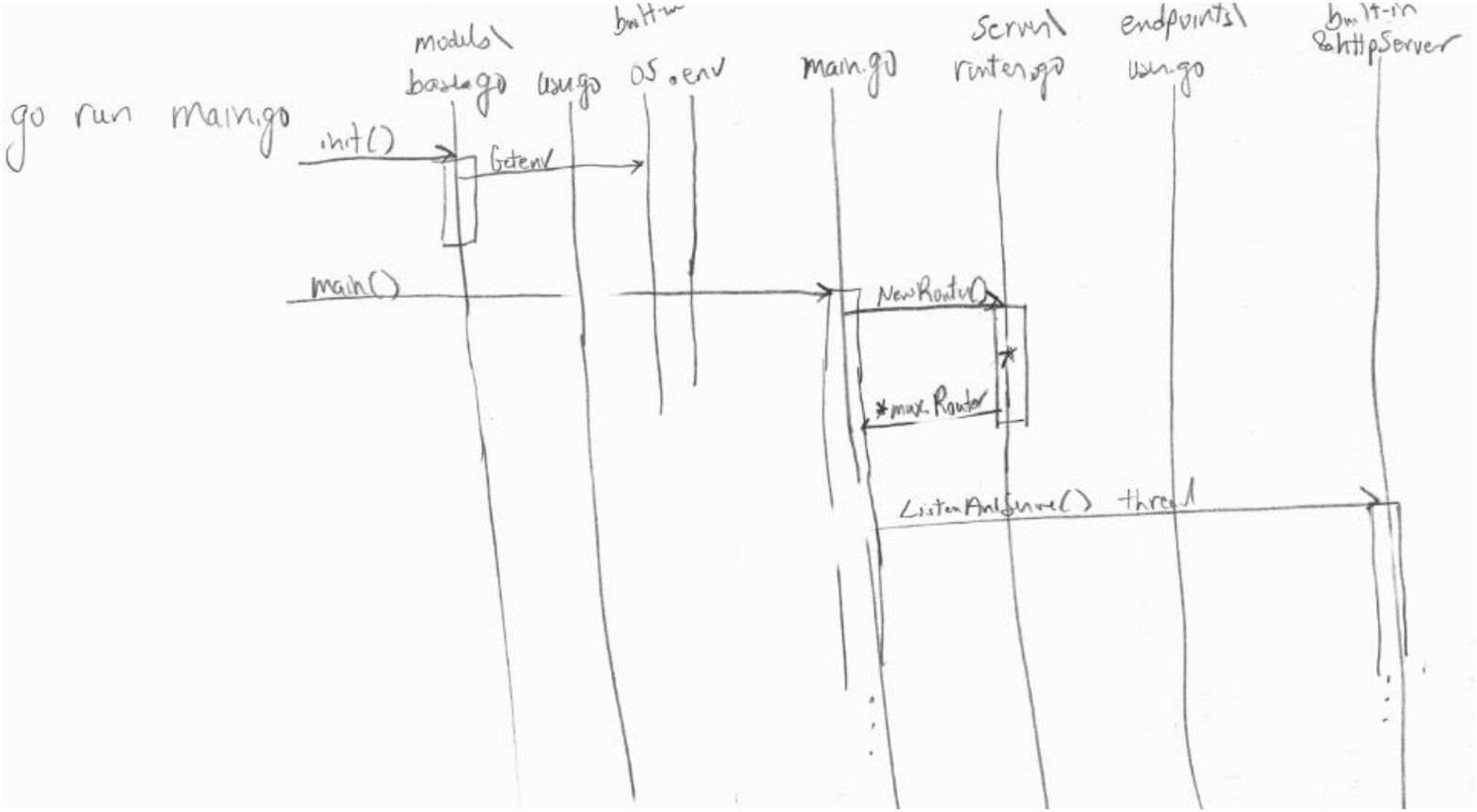
```
// Get the db connection URL from the environment.
    username := os.Getenv("db_user")
    password := os.Getenv("db_passwd")
    dbHost := os.Getenv("db_host")
    dbName := os.Getenv("db_name")
    dbType := os.Getenv("db_type")

    dbURI := fmt.Sprintf("%s:%s@(%s)/%s?charset=utf8",
        username, password, dbHost, dbName)
    fmt.Println(dbURI)
```

Server Not Instantiated Due to DB

```
conn, err := gorm.Open(dbType, dbURI)
    if err != nil {
        fmt.Println(err)                    // Line 40
    }
    db = conn

    err = db.DB().Ping()
    if err != nil {
        panic(err)                          // Line 46
    }
}
```

Setup Server

Sequence Diagram



Environment Variables For Local DB

The environment variables need to match how you set up a local database

Example:

db_name = about_test

db_user = web_user

db_passwd = resu_bew

db_type = mysql

db_host = localhost

Server Running

Example of server running on Windows machines:

```
AbOut\backend> go run main.go
web_user:resu_bew@(localhost)/abOut_test?charset=utf8
```

To see the network status:
netstat is a utility that displays network connections for TCP

> netstat -n
–n displays addresses and port numbers in numerical form

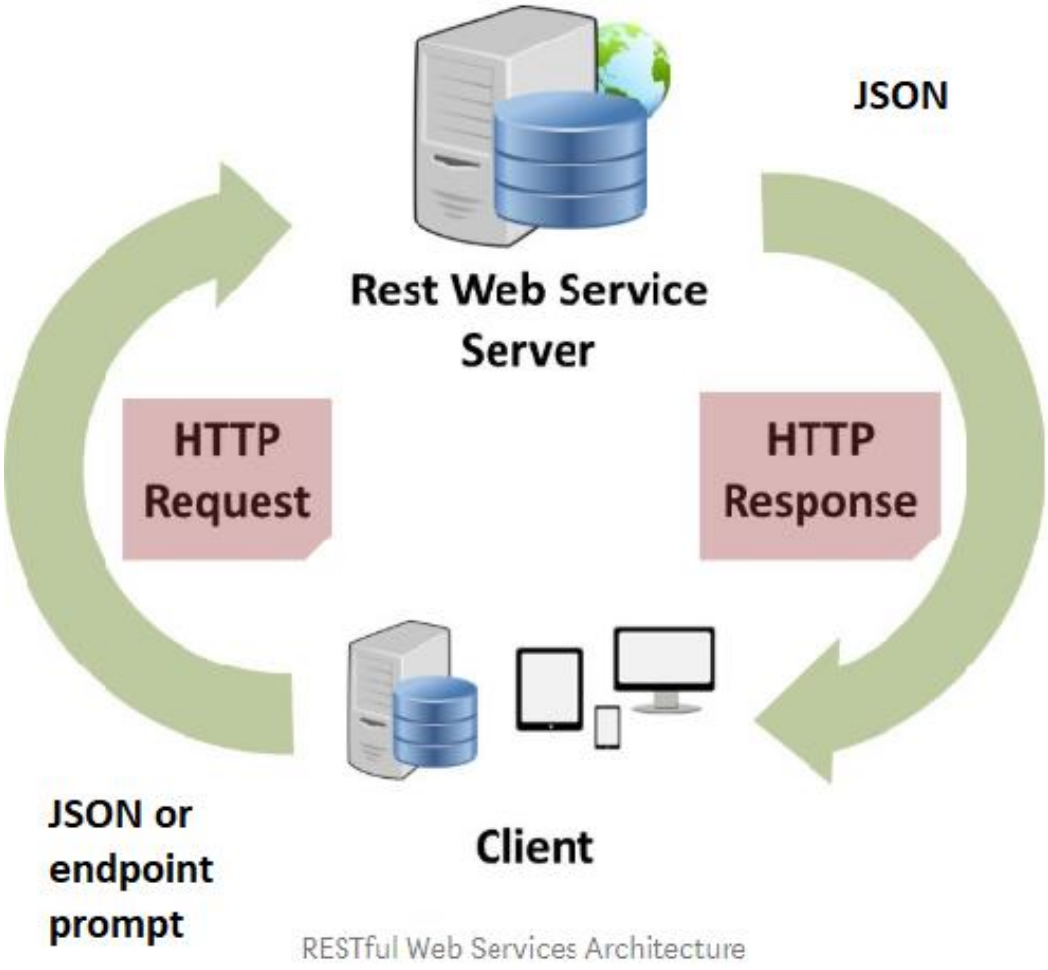
Example Network Status

```
PS C:\1Work\AB_Courses\ESOF_322\ESOF322---Current\AbOut_Refactor\sprint2_wantToMergeEditUserInfo\
AbOut> netstat -n

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    10.38.32.39:55825      52.230.222.68:443      ESTABLISHED
  TCP    10.38.32.39:55826      162.220.222.168:443    ESTABLISHED
  TCP    10.38.32.39:55866      10.33.93.13:445        ESTABLISHED
  TCP    10.38.32.39:55906      40.97.162.162:443      ESTABLISHED
  TCP    10.38.32.39:55933      162.125.1.3:443        CLOSE_WAIT
  TCP    10.38.32.39:55974      3.231.46.251:443       CLOSE_WAIT
  TCP    10.38.32.39:56048      108.177.98.188:5228    ESTABLISHED
```

...

```
  TCP    127.0.0.1:49801        127.0.0.1:49802        ESTABLISHED
  TCP    127.0.0.1:49802        127.0.0.1:49801        ESTABLISHED
  TCP    127.0.0.1:55880        127.0.0.1:5939         ESTABLISHED
  TCP    127.0.0.1:55883        127.0.0.1:55884        ESTABLISHED
  TCP    127.0.0.1:55884        127.0.0.1:55883        ESTABLISHED
  TCP    127.0.0.1:55922        127.0.0.1:55923        ESTABLISHED
  TCP    127.0.0.1:55923        127.0.0.1:55922        ESTABLISHED
  TCP    127.0.0.1:55934        127.0.0.1:55935        ESTABLISHED
  TCP    127.0.0.1:55935        127.0.0.1:55934        ESTABLISHED
  TCP    [::1]:3306             [::1]:56501            ESTABLISHED
  TCP    [::1]:56501            [::1]:3306             ESTABLISHED
```

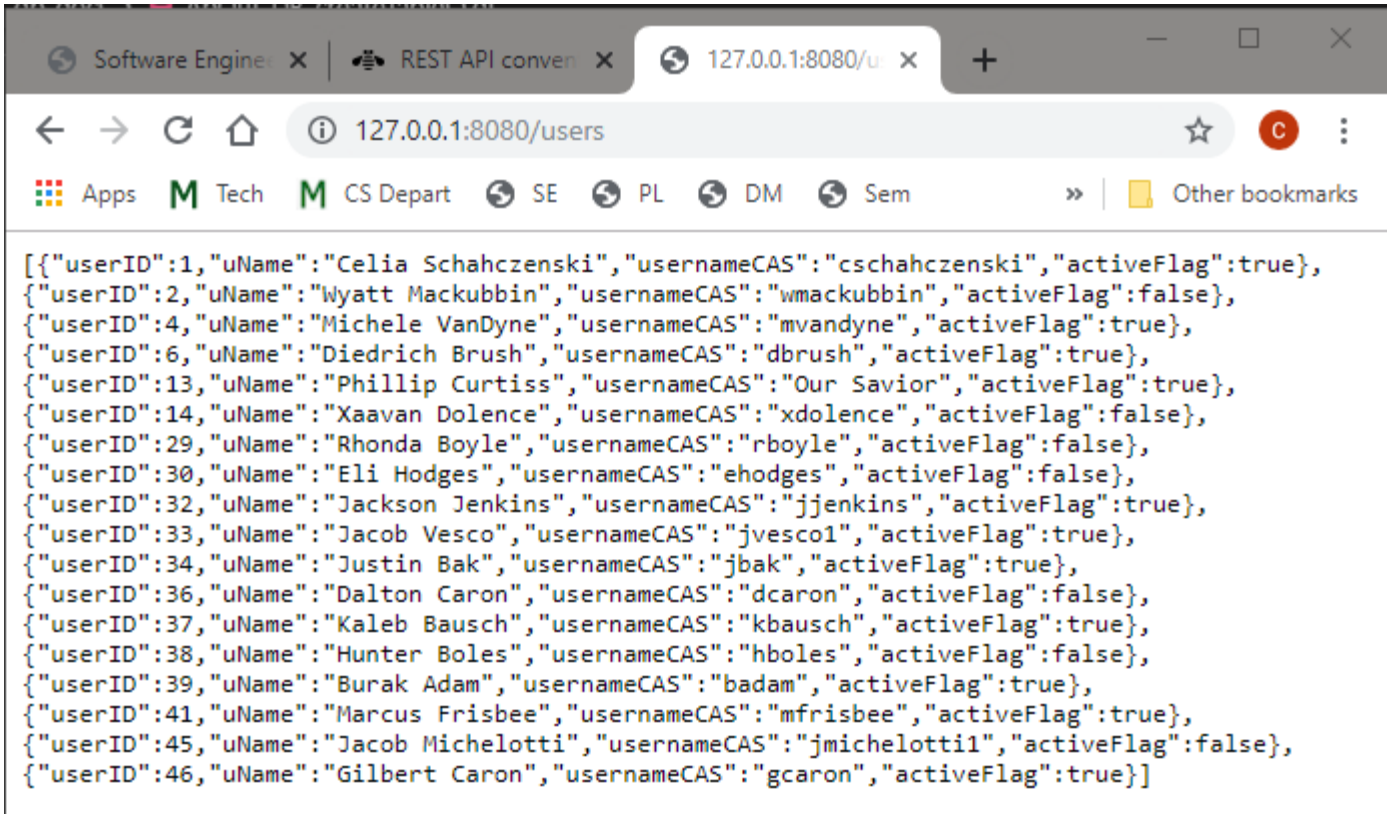
JSON



RESTful Web Services Architecture

AbOut Web Service Examples

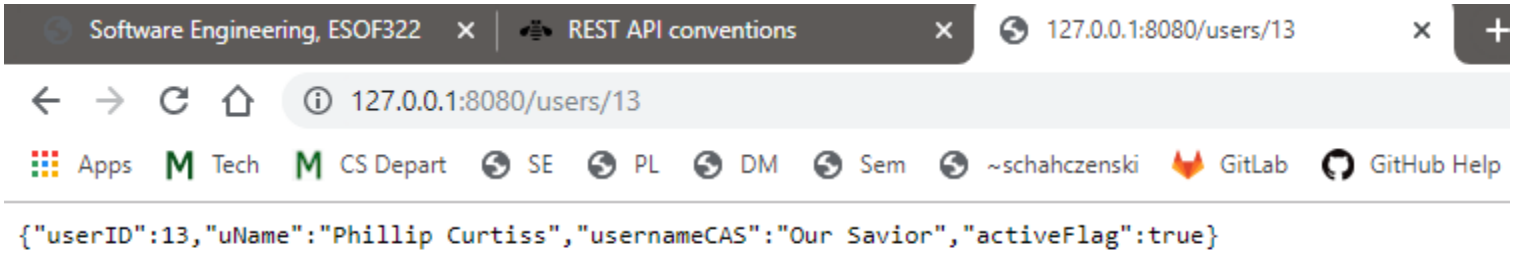
In Google Chrome:




```
[{"userID":1,"uName":"Celia Schahczenski","usernameCAS":"cschahczenski","activeFlag":true},
{"userID":2,"uName":"Wyatt Mackubbin","usernameCAS":"wmackubbin","activeFlag":false},
{"userID":4,"uName":"Michele VanDyne","usernameCAS":"mvandyne","activeFlag":true},
{"userID":6,"uName":"Diedrich Brush","usernameCAS":"dbrush","activeFlag":true},
{"userID":13,"uName":"Phillip Curtiss","usernameCAS":"Our Savior","activeFlag":true},
{"userID":14,"uName":"Xaavan Dolence","usernameCAS":"xdolence","activeFlag":false},
{"userID":29,"uName":"Rhonda Boyle","usernameCAS":"rboyle","activeFlag":false},
{"userID":30,"uName":"Eli Hodges","usernameCAS":"ehodges","activeFlag":false},
{"userID":32,"uName":"Jackson Jenkins","usernameCAS":"jjenkins","activeFlag":true},
{"userID":33,"uName":"Jacob Vesco","usernameCAS":"jvesco1","activeFlag":true},
{"userID":34,"uName":"Justin Bak","usernameCAS":"jbak","activeFlag":true},
{"userID":36,"uName":"Dalton Caron","usernameCAS":"dcaron","activeFlag":false},
{"userID":37,"uName":"Kaleb Bausch","usernameCAS":"kbausch","activeFlag":true},
{"userID":38,"uName":"Hunter Boles","usernameCAS":"hboles","activeFlag":false},
{"userID":39,"uName":"Burak Adam","usernameCAS":"badam","activeFlag":true},
{"userID":41,"uName":"Marcus Frisbee","usernameCAS":"mfrisbee","activeFlag":true},
{"userID":45,"uName":"Jacob Michelotti","usernameCAS":"jmichelotti1","activeFlag":false},
{"userID":46,"uName":"Gilbert Caron","usernameCAS":"gcaron","activeFlag":true}]
```

AbOut Web Service Examples

In Google Chrome:



GORM

From GoLang Tutorial https://mindbowser.com › golang-go-with-gorm-2

> The GORM is fantastic ORM library for Golang, aims to be developer friendly. It is an ORM library for dealing with relational databases.
> This gorm library is developed on the top of database/sql package.

ORM – Object-Relational Mapping

GORM – Connection to DB

Recall error when db connection was wrong, error occurred in backend/models/base.go

```
// Init sets up the database connection.
func init() {
                .
                .
                .
 conn, err := gorm.Open(dbType, dbURI)
    if err != nil {
        fmt.Println(err)                    // Line 40
    }
    db = conn

 err = db.DB().Ping()
    if err != nil {
        panic(err)                          // Line 46
    }
}
```

GORM Example

In models/user.go

```
// User contains all of the information associated with a user.
// Not a fan of these variable names, but will be consistent with db attributes.
type User struct {

        UserID      int
`gorm:"column:userId;type:int(11);PRIMARY_KEY;AUTO_INCREMENT;NOT NULL"
json:"userID"`

        UName       string `gorm:"column:uName;type:varchar(50);NOT NULL"
json:"uName"`

        UsernameCAS string
`gorm:"column:usernameCAS;type:varchar(50);NOT NULL"
json:"usernameCAS"`

        ActiveFlag  bool   `gorm:"column:activeFlag;type:tinyint(1);DEFAULT
NULL" json:"activeFlag"`
}
```

GORM Example

In models/user.go

```
// GetUser returns the single user with userID from the database.
func (UserRepo) GetUser(userID int) (User, error) {
	user := User{UserID: userID}
	if err := db.Where("userID = ?", userID).First(&user).Error; err != nil {
		errString := fmt.Sprintf("failed to fetch user from the database: %s", err)
		return User{}, errors.New(errString)
	}
	return user, nil
}
```

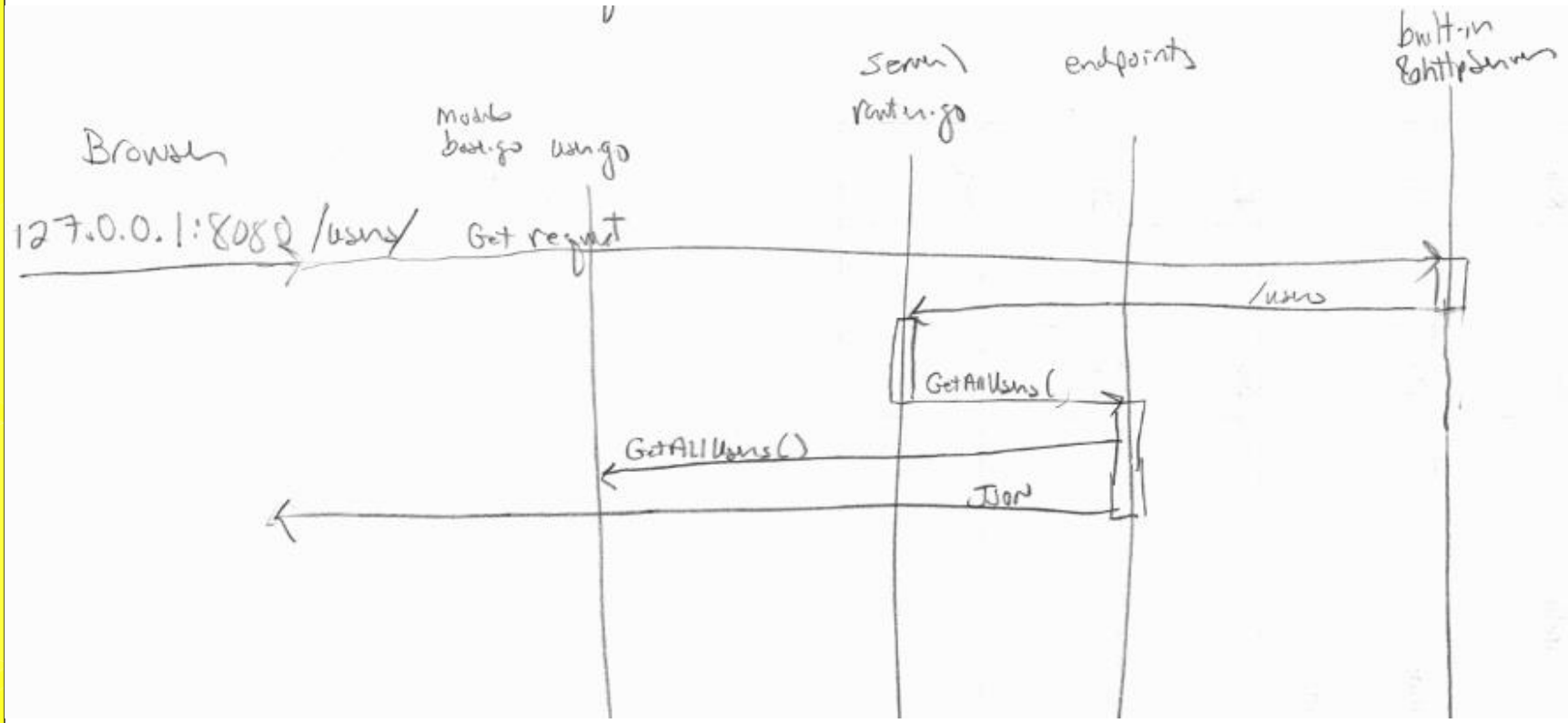
GORM Example

In models/user.go

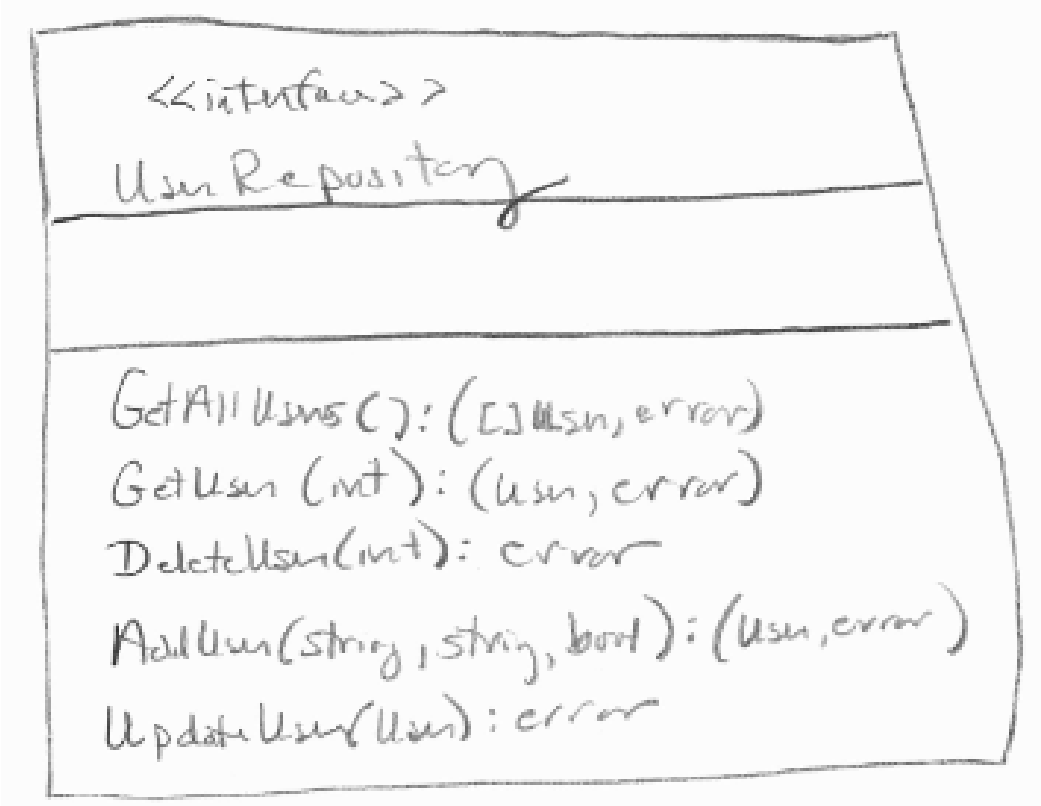
```
// GetAllUsers returns all users from the database as a list.
func (UserRepo) GetAllUsers() ([]User, error) {
	users := []User{}
	if err := db.Select("userId, uName, usernameCAS, activeFlag").Order("userId").Find(&users).Error; err != nil {
		return nil, err
	}
	return users, nil
}
```

Request Endpoint

Sequence Diagram



models/user.go



Structures

Class Diagrams