HW15: Final Project database
Anu Srikanth

Now URL: https://final-proj-mlolscbxye.now.sh

Introduction

The first two of my A-level Use cases have to do with the User or Account model. User and account refer to the same model. They are used to associate each of the users with their specific user account details, tickets they have purchased, ticket information, etc. This is the reason I have chosen to complete this model first.

A-Level Use Cases		
Use Case Name	**Actor(s)**	**Description**
Create Account	User	The user can create an account with credentials
Login	User	The user should be able to login with previously created credentials

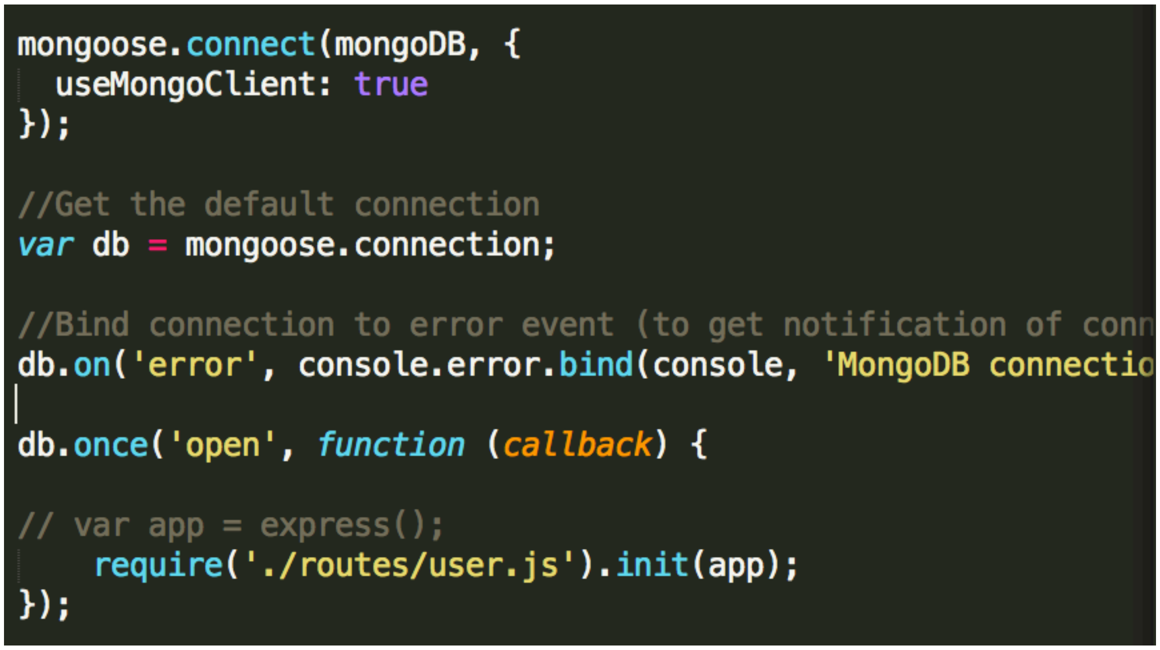
Some of the modules/packages that I have added to my project are as follows. These were not included in the initial project proposal.

Modules and Packages		
Package/Module Name	**Component**	**Description**
mongoose	Database	This allows for easy schema creation for models that give them a more definite structure and makes it easier for testing and validations.

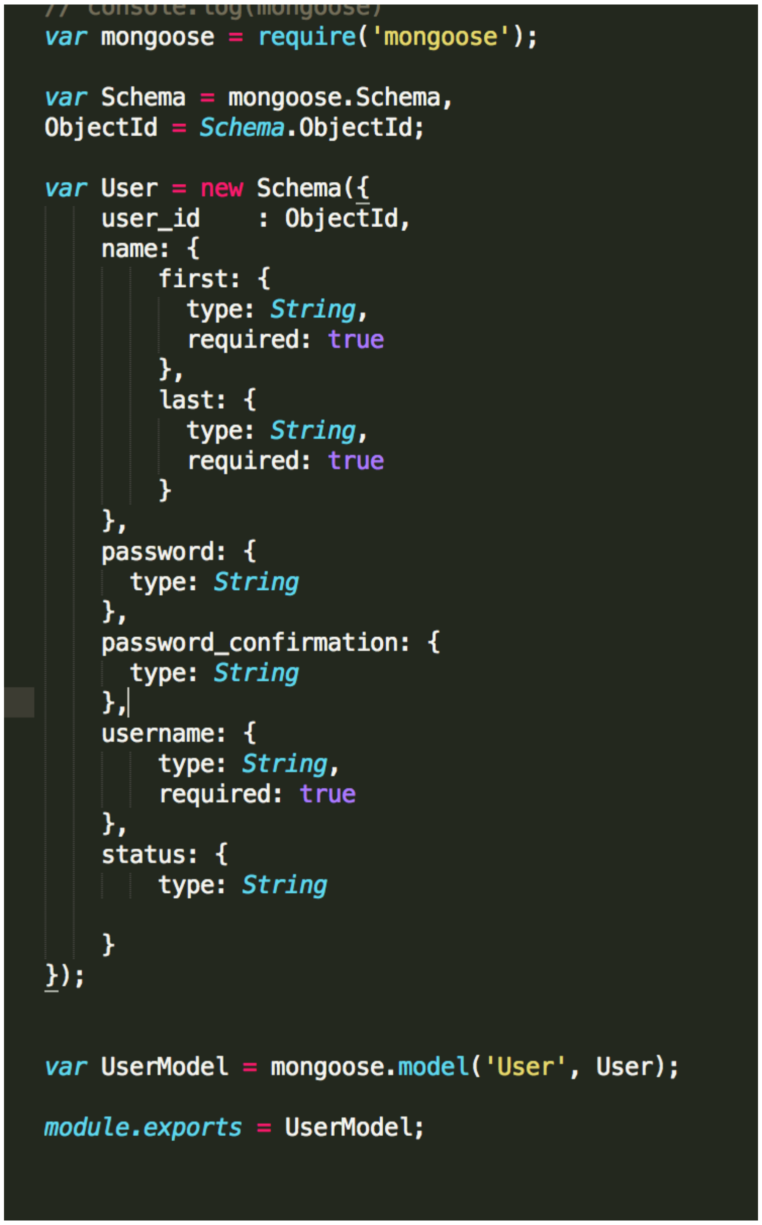
Mocha	Testing	JavaScript framework to make asynchronous testing easier
Chai	Testing	Assertions library

Model functionality

The following code sample is from my user model, the model that Ive chosen to store in the database. Here I am connecting to the mongodb database on mlab with credentials that I have previously provided. Then I set up my models using schema functionality provided by mongoose that helps keep models in a particular structure.

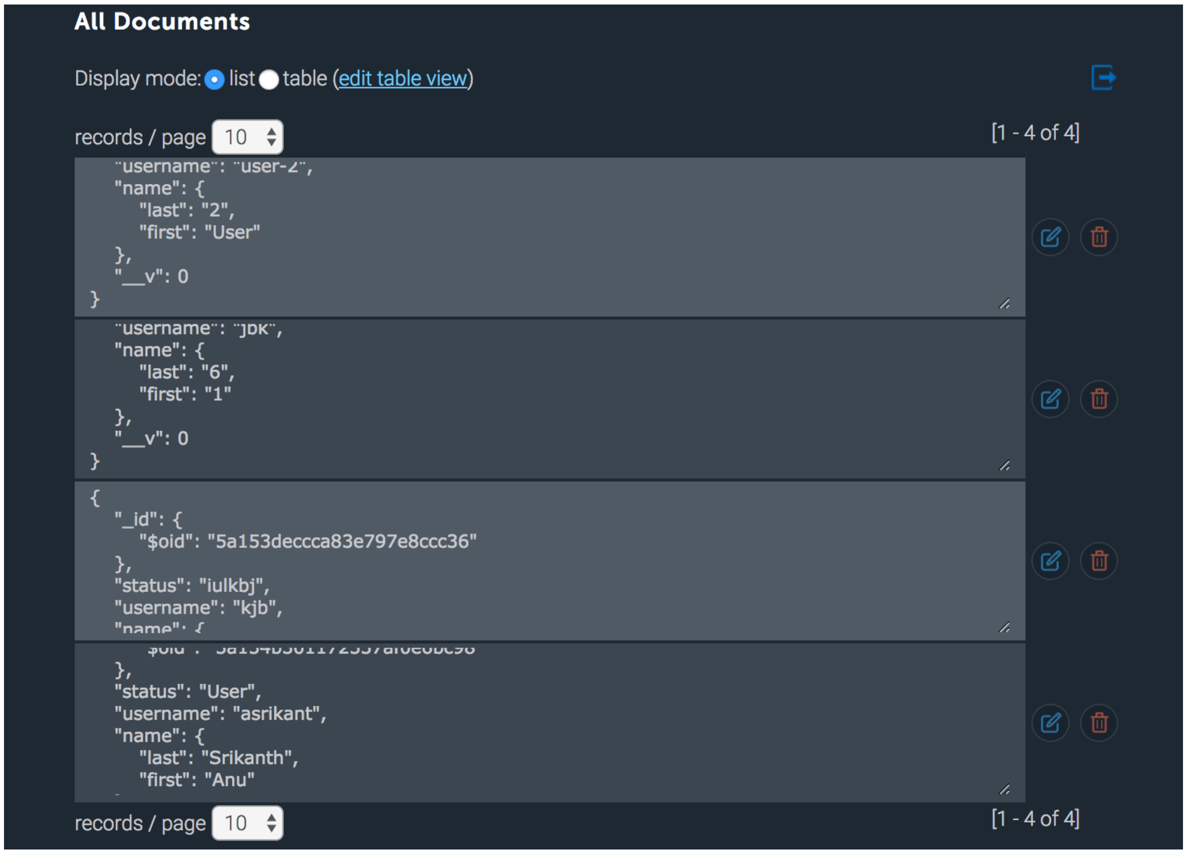


```
mongoose.connect(mongoDB, {
  useMongoClient: true
});

//Get the default connection
var db = mongoose.connection;

//Bind connection to error event (to get notification of conn
db.on('error', console.error.bind(console, 'MongoDB connectio

db.once('open', function (callback) {

// var app = express();
    require('./routes/user.js').init(app);
});
```



```
// console.log(mongoose)
var mongoose = require('mongoose');

var Schema = mongoose.Schema,
ObjectId = Schema.ObjectId;

var User = new Schema({
    user_id    : ObjectId,
    name: {
        first: {
          type: String,
          required: true
        },
        last: {
          type: String,
          required: true
        }
    },
    password: {
      type: String
    },
    password_confirmation: {
      type: String
    },
    username: {
        type: String,
        required: true
    },
    status: {
        type: String

    }
});


var UserModel = mongoose.model('User', User);

module.exports = UserModel;
```

Documents updated/added in mlab's database showing a successful connection



```
    "username": "user-2",
    "name": {
        "last": "2",
        "first": "User"
    },
    "__v": 0
}
```

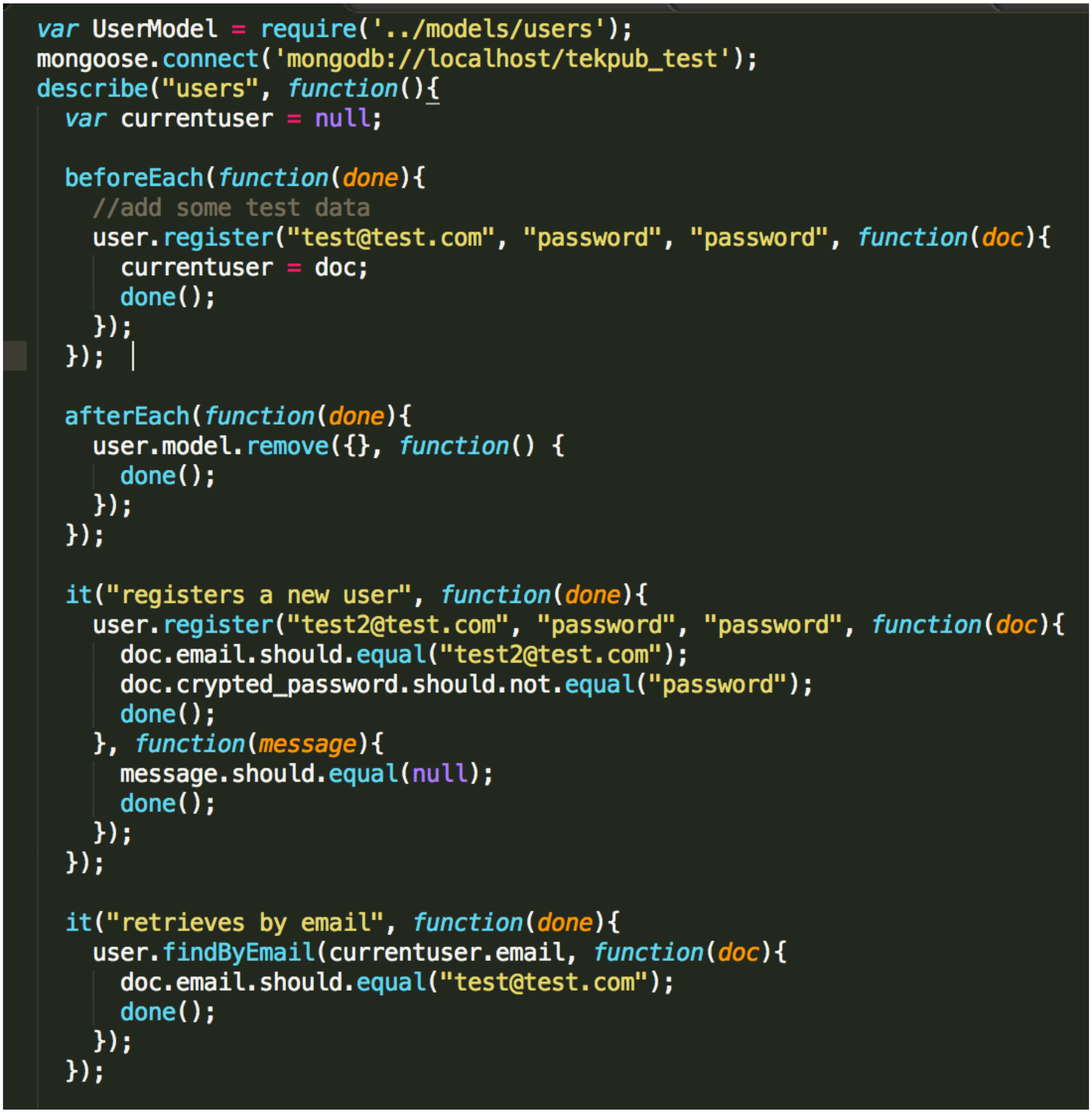
```
    "username": "jbk",
    "name": {
        "last": "6",
        "first": "1"
    },
    "__v": 0
}
```

```
{
    "_id": {
        "$oid": "5a153deccca83e797e8ccc36"
    },
    "status": "iulkbj",
    "username": "kjb",
    "name": {
```

```
        "$oid": "5a154b501172557af0e6bc98"
    },
    "status": "User",
    "username": "asrikant",
    "name": {
        "last": "Srikanth",
        "first": "Anu"
```

Model Testing

Below is some tests that I have written using the Mocha testing framework and a local database in order to not interfere with the data on the online one and also to speed up testing. They have passed.



```
var UserModel = require('../models/users');
mongoose.connect('mongodb://localhost/tekpub_test');
describe("users", function(){
  var currentuser = null;

  beforeEach(function(done){
    //add some test data
    user.register("test@test.com", "password", "password", function(doc){
      currentuser = doc;
      done();
    });
  });

  afterEach(function(done){
    user.model.remove({}, function() {
      done();
    });
  });

  it("registers a new user", function(done){
    user.register("test2@test.com", "password", "password", function(doc){
      doc.email.should.equal("test2@test.com");
      doc.crypted_password.should.not.equal("password");
      done();
    }, function(message){
      message.should.equal(null);
      done();
    });
  });

  it("retrieves by email", function(done){
    user.findByEmail(currentuser.email, function(doc){
      doc.email.should.equal("test@test.com");
      done();
    });
  });
```

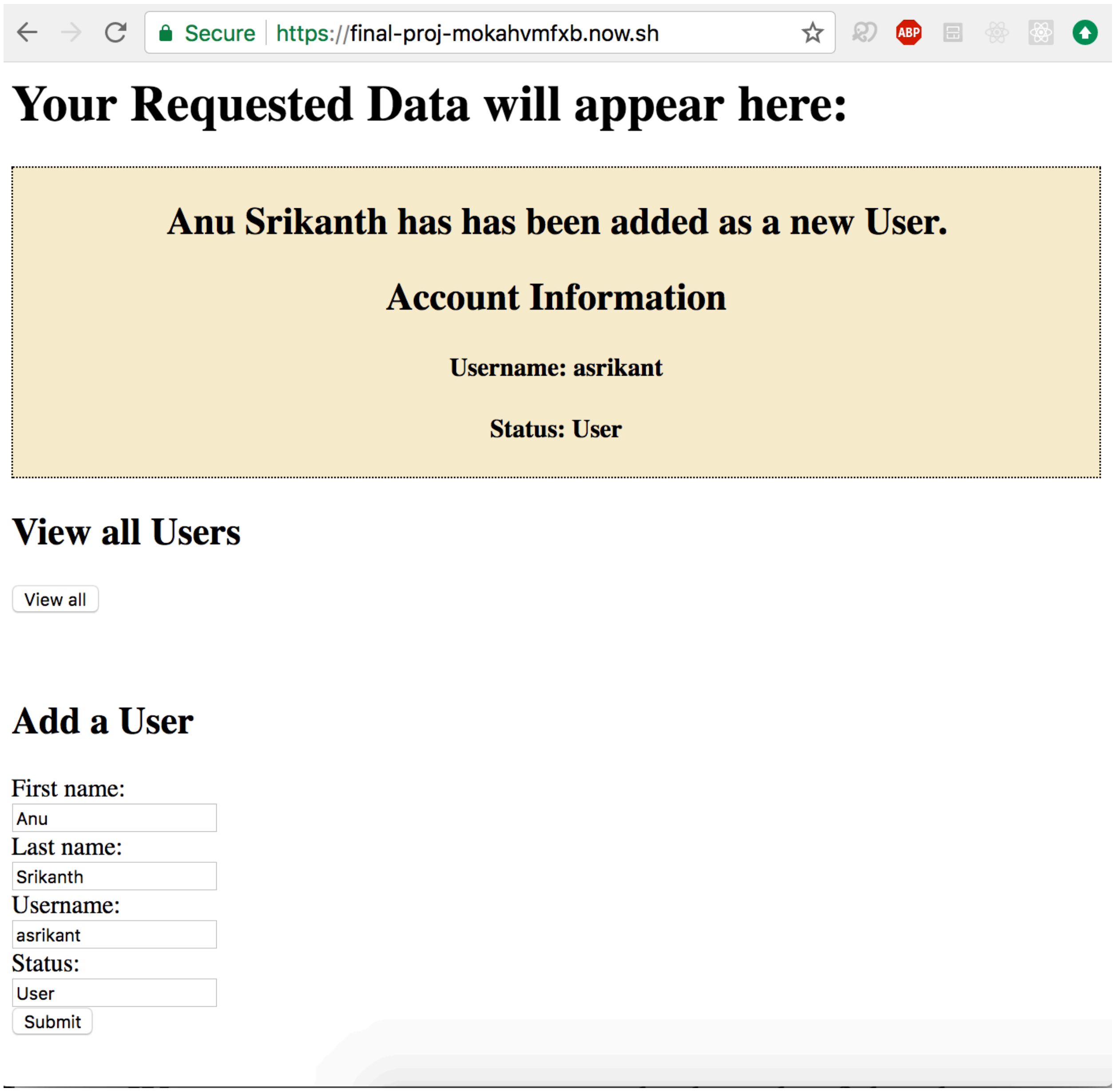
<u>CRUD functionality</u>

The following is how the process of executing the CRUD functions work on the backend. This will be followed up by working examples.

- **a.** Form on public index page triggers an ajax request
- **b.** Ajax request sent to routes along with the associated method
- **c.** Routes file enumerates the different urls and paths that could be passed to it and does the relevant function when the ajax request url matches a route.
- **d.** This function updates/retrieved information in the database and follows up with a success or an error response

i. If there is an error response, an error handler is called that logs the error and responds back to the client
ii. If there is a success response, the relevant view is populated with information and then displayed on the browser.

1. CREATE:


Secure https://final-proj-mokahvmfxb.now.sh

Your Requested Data will appear here:

Anu Srikanth has has been added as a new User.

Account Information

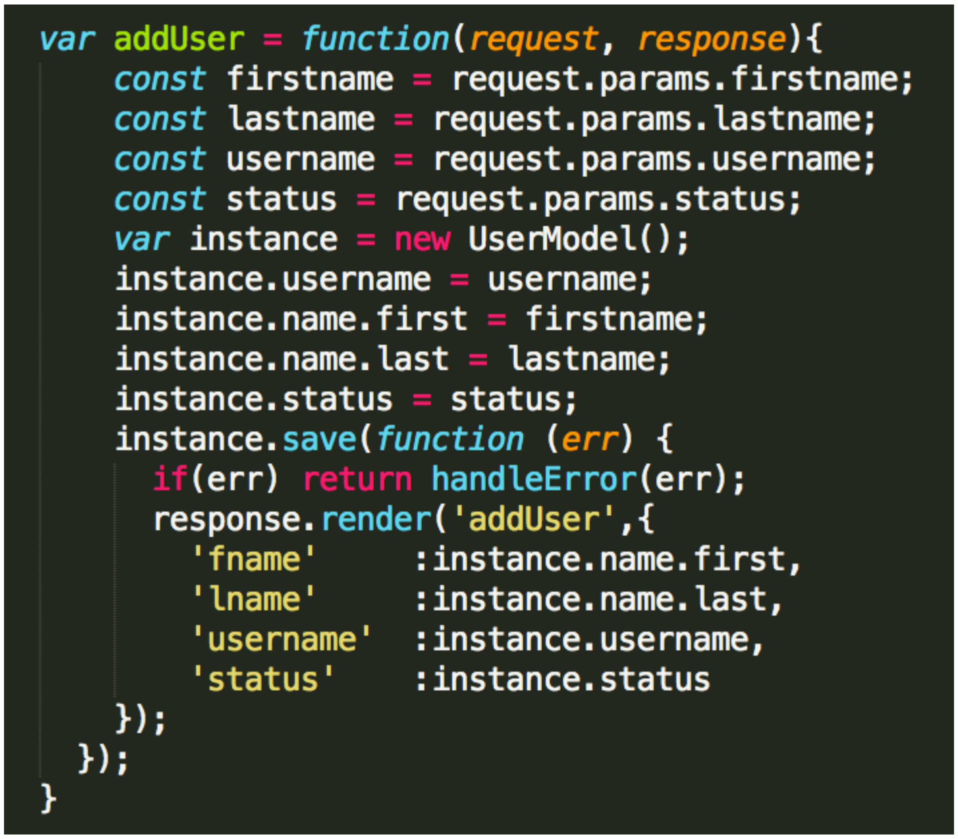
Username: asrikant

Status: User

View all Users

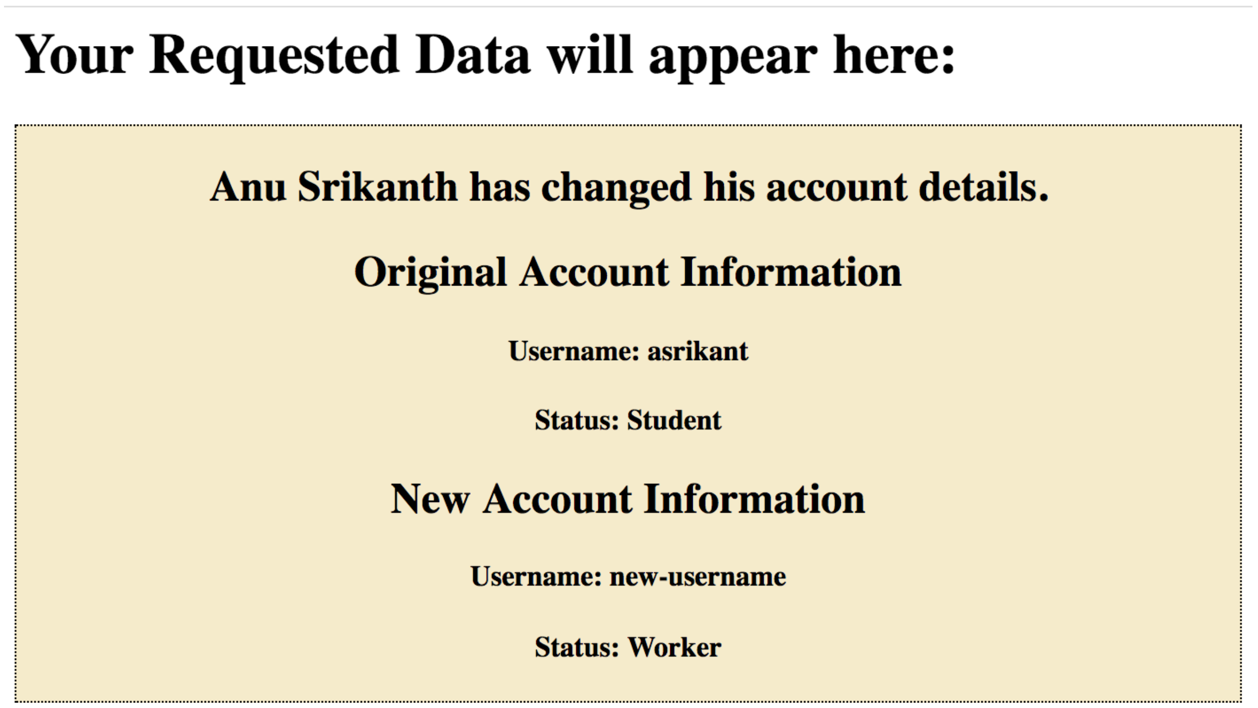
Add a User

First name: Anu
Last name: Srikanth
Username: asrikant
Status: User



```
var addUser = function(request, response){
    const firstname = request.params.firstname;
    const lastname = request.params.lastname;
    const username = request.params.username;
    const status = request.params.status;
    var instance = new UserModel();
    instance.username = username;
    instance.name.first = firstname;
    instance.name.last = lastname;
    instance.status = status;
    instance.save(function (err) {
      if(err) return handleError(err);
      response.render('addUser',{
        'fname'      :instance.name.first,
        'lname'      :instance.name.last,
        'username'   :instance.username,
        'status'     :instance.status
    });
  });
}
```

2. UPDATE



Your Requested Data will appear here:

Anu Srikanth has changed his account details.

Original Account Information

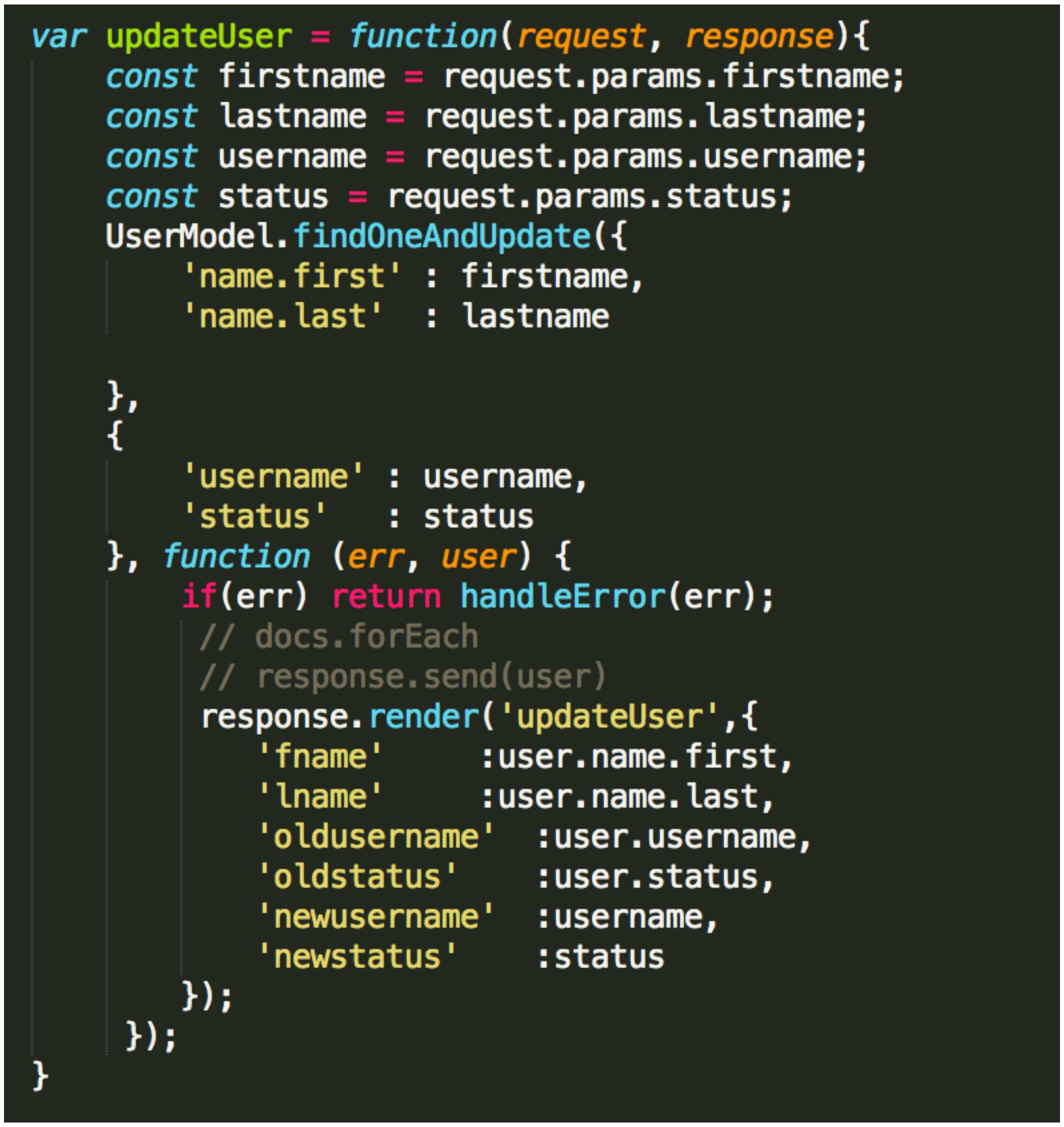
Username: asrikant

Status: Student

New Account Information

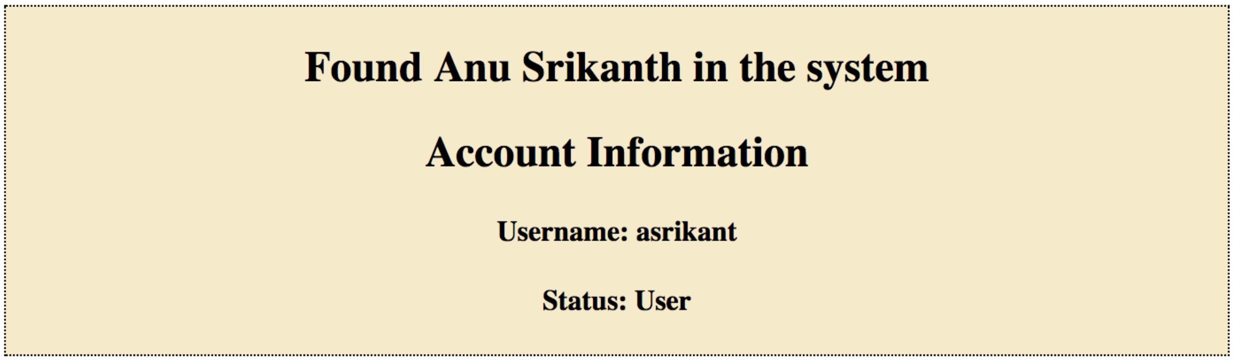
Username: new-username

Status: Worker



```
var updateUser = function(request, response){
    const firstname = request.params.firstname;
    const lastname = request.params.lastname;
    const username = request.params.username;
    const status = request.params.status;
    UserModel.findOneAndUpdate({
        'name.first' : firstname,
        'name.last'  : lastname

    },
    {
        'username' : username,
        'status'   : status
    }, function (err, user) {
        if(err) return handleError(err);
         // docs.forEach
         // response.send(user)
         response.render('updateUser',{
            'fname'      :user.name.first,
            'lname'      :user.name.last,
            'oldusername'   :user.username,
            'oldstatus'     :user.status,
            'newusername'   :username,
            'newstatus'     :status
        });
     });
}
```

3. RETRIEVE

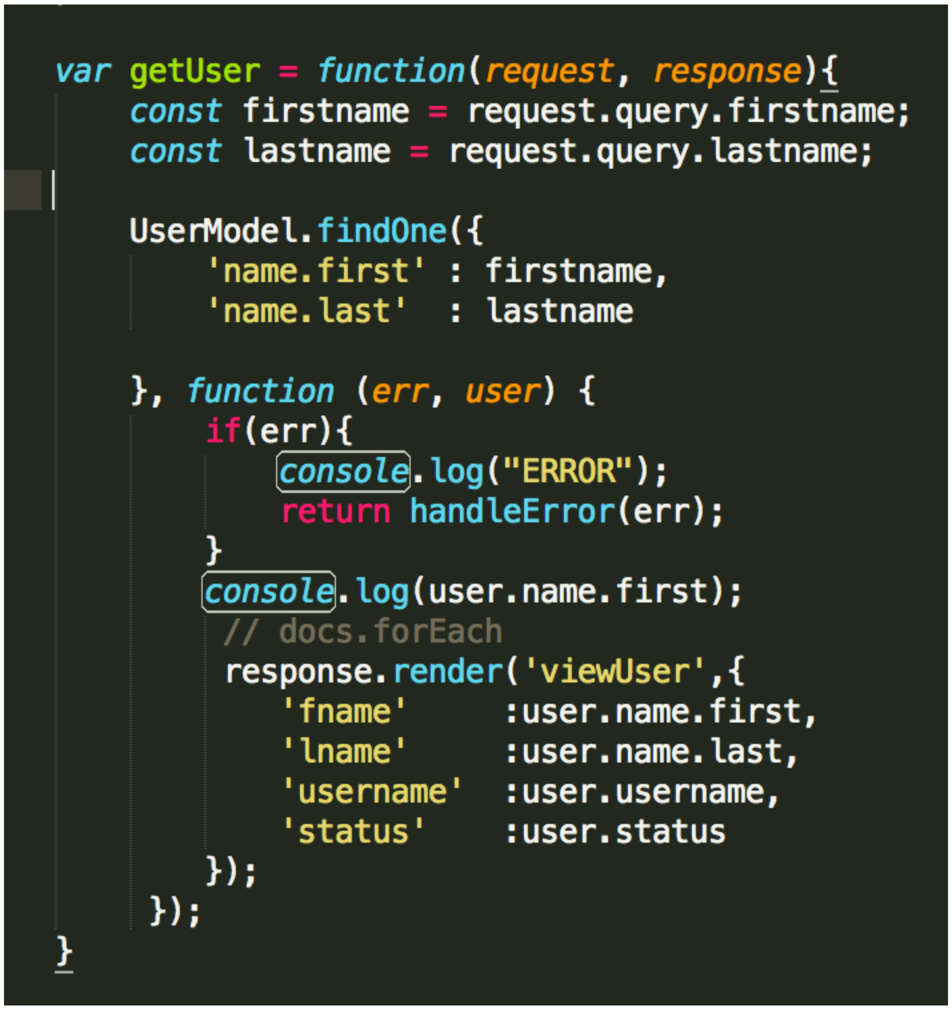


Found Anu Srikanth in the system

Account Information

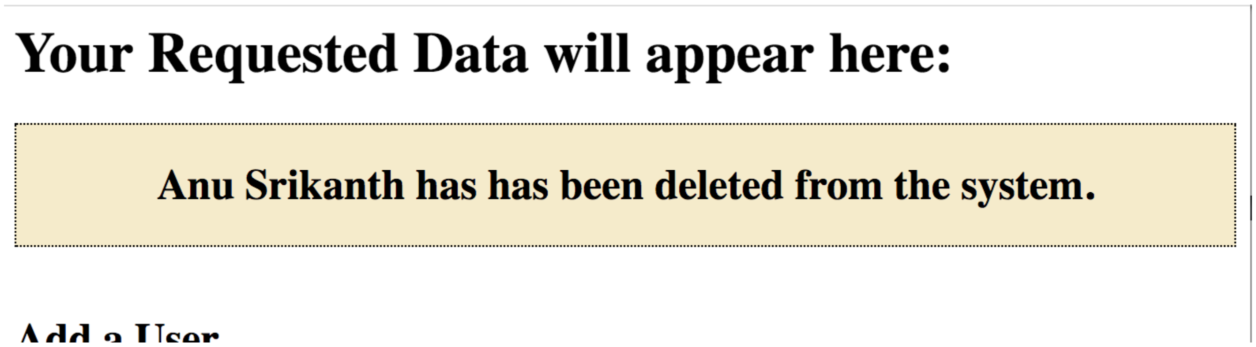
Username: asrikant

Status: User



```
var getUser = function(request, response){
    const firstname = request.query.firstname;
    const lastname = request.query.lastname;

    UserModel.findOne({
        'name.first' : firstname,
        'name.last'  : lastname

    }, function (err, user) {
        if(err){
            console.log("ERROR");
            return handleError(err);
        }
        console.log(user.name.first);
         // docs.forEach
         response.render('viewUser',{
            'fname'      :user.name.first,
            'lname'      :user.name.last,
            'username'   :user.username,
            'status'     :user.status
        });
     });
}
```

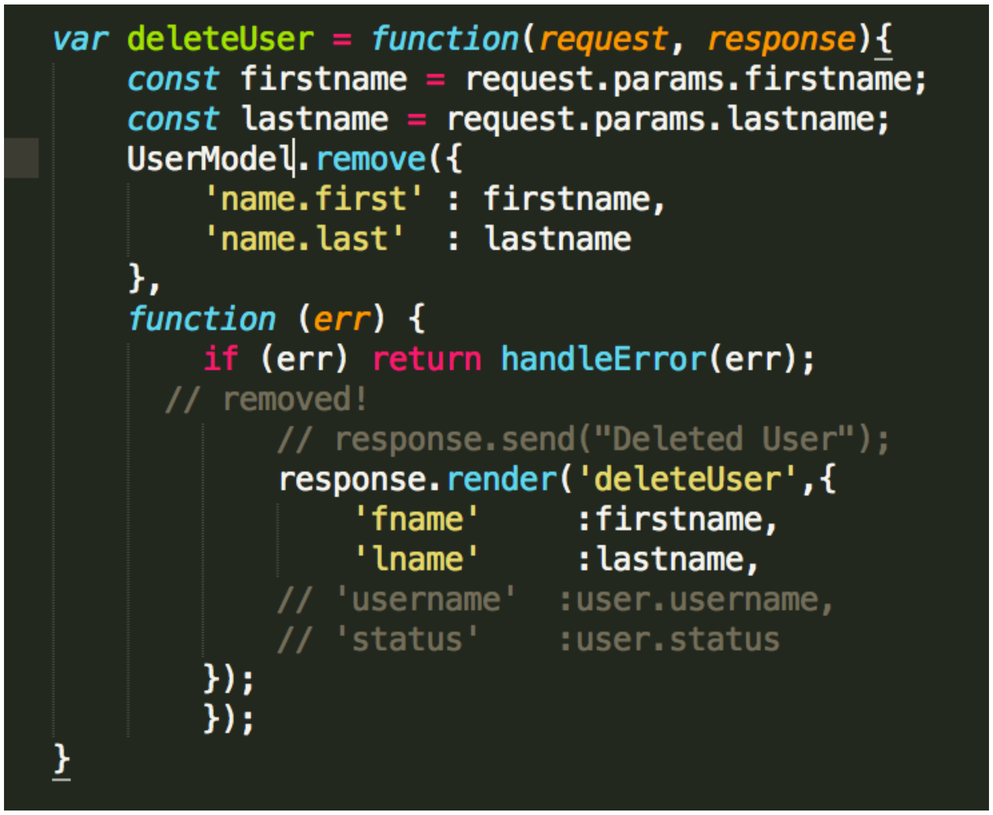
4. DELETE



Your Requested Data will appear here:

Anu Srikanth has has been deleted from the system.

Add a User



```
var deleteUser = function(request, response){
    const firstname = request.params.firstname;
    const lastname = request.params.lastname;
    UserModel.remove({
        'name.first' : firstname,
        'name.last'  : lastname
    },
    function (err) {
        if (err) return handleError(err);
      // removed!
          // response.send("Deleted User");
          response.render('deleteUser',{
              'fname'     :firstname,
              'lname'     :lastname,
          // 'username'  :user.username,
          // 'status'    :user.status
        });
        });
}
```