Final Project Report: Nail

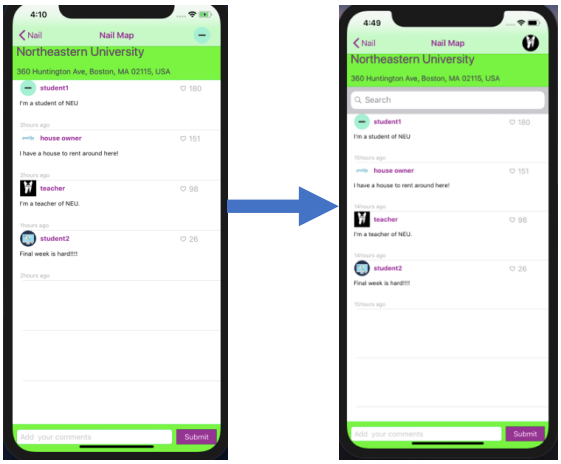
Zhaoyu Yan

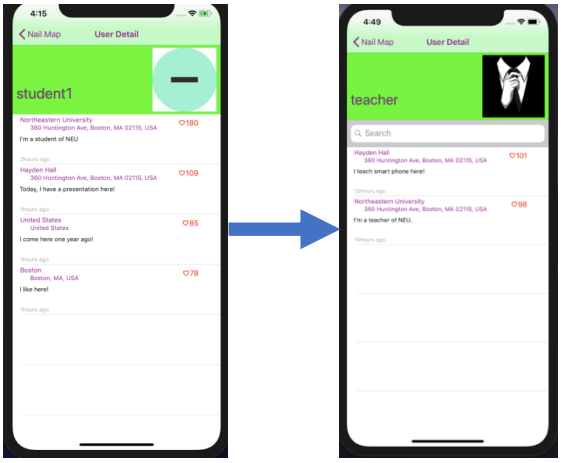
1. Project improving

a. During my demo, I found the new user by signing up was denied by login page, while I tested it successfully yesterday. Now, I realize it' s due to response time which is faster when app and server in the same network, so when I presented the new user have not finished uploading data when login page get all users. And now, I use asynchronized task to fix it.

```
DispatchQueue.main.async {
    LoginViewController.users.append(user)
}
```

b. TA mentions I should add search bar to comments. Following is what I do:





2. Introduction

   There are many social apps and many apps based on location as well, but social apps all rely on relationships among people and apps based on location are only focus on one specific aspect: food, weather or rent. I combine two kinds of apps to create my app, Nail. Nail is an app that can write comments to any place by anyone, so it’ s like a social app based on map. As you see, Nail is used to share information in a place or an area.

3. Skills or tools I used for my project

   a. Google Map API & Google Places API

   b. Pull-up nested view controller

   c. build my own server by mongoDB and node.js

4. Go through my project with codes

   a. My project has two core entities: User, Comment. One sub entity:

Photo, is for easy to decode from json file.

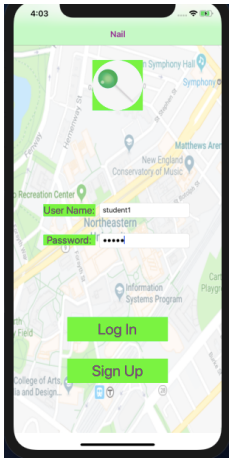
nail 〉 nail 〉 entity 〉 User.swift 〉 User

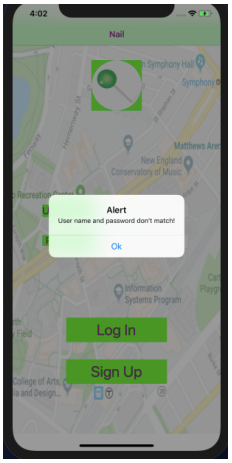
```swift
//
//  User.swift
//  nail
//
//  Created by Zhaoyu Yan on 12/11/18.
//  Copyright © 2018 Zhaoyu Yan. All rights reserved.
//

import Foundation

struct User: Decodable{
    var _id:String
    let username:String
    var password:String
    var photo: Photo

}
```

nail 〉 nail 〉 entity 〉 Comment.swift 〉 Comment

```swift
//
//  Comment.swift
//  nail
//
//  Created by Zhaoyu Yan on 12/13/18.
//  Copyright © 2018 Zhaoyu Yan. All rights reserved.
//

import Foundation

struct Comment: Decodable{
    var _id:String
    var username:String
    var userPhoto:Photo
    var placeId:String
    var placeName:String
    var placeAddress:String
    var content:String
    var publishDate:String
    var likes: Int
}
```

b. LoginViewController is for getting all users, users log in with validation and able to guide to SignupViewController.

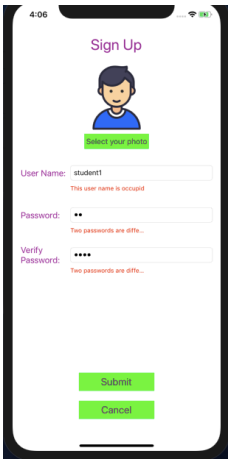





```swift
func getUsers(){
    let jsonUrlStr = LoginViewController.serverUrl+"/users"
    guard let jsonUrl = URL(string: jsonUrlStr) else {return}

    URLSession.shared.dataTask(with: jsonUrl){ (data, response, err) in
        guard let data = data else {return}
        do{
            LoginViewController.users = try JSONDecoder().decode([User].self, from: data)
            print("Get users successfully!")
            //print(user.photo)
        }catch let jsonErr{
            print("Error serializing json:", jsonErr)
        }
        }.resume()
}
```

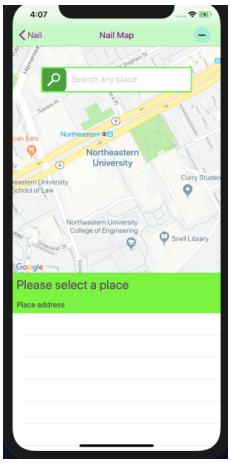
c. SignupViewController is for create a new user and upload to server.

Username should be unique, and two passwords should be same.

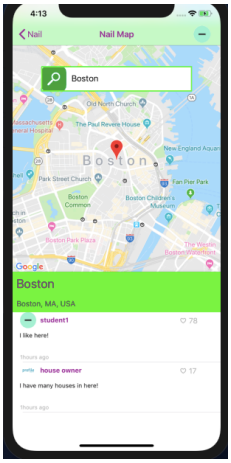



```
func upLoadUser(username:String, password: String, photo:Data){
    let jsonUrlStr = LoginViewController.serverUrl+"/users"
    guard let jsonUrl = URL(string: jsonUrlStr) else {return}
    var request = URLRequest(url:jsonUrl)
    request.httpMethod = "POST"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    let photoDataArr = [UInt8](photo)
    let postJson: [String: Any] = ["username": username, "password": password, "photo":photoDataArr]
    let postData = try? JSONSerialization.data(withJSONObject: postJson)
    //print(postData!)
    request.httpBody = postData

    URLSession.shared.dataTask(with: request){ (data, response, err) in
        //let httpResponse = response as? HTTPURLResponse
        //print(httpResponse!.statusCode)
        guard let data = data else {return}
        do{
            let user = try JSONDecoder().decode(User.self, from: data)
            print(user.username)
            DispatchQueue.main.async {
                LoginViewController.users.append(user)
            }
        }catch let jsonErr{
            print("Error serializing json:", jsonErr)
        }
    }.resume()
}
```

d. MainViewController is for presenting a google map and can search locations.

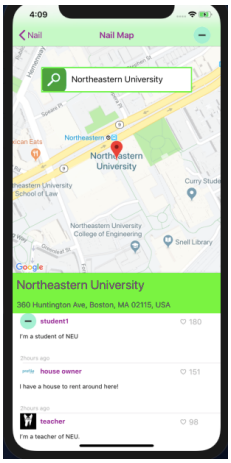


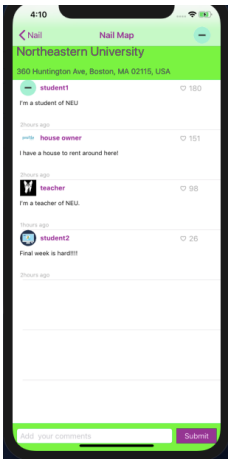

```
func initMapView(){
    let camera = GMSCameraPosition.camera(withLatitude: 42.339753, longitude: -71.089088, zoom: 17)
    mapView = GMSMapView.map(withFrame: CGRect(x: 0, y: 0, width: view.bounds.width, height: view.bounds.height-350), camera: camera)
    view.addSubview(mapView)
}
```




```
func viewController(_ viewController: GMSAutocompleteViewController, didAutocompleteWith place: GMSPlace) {
    let lat = place.coordinate.latitude
    let long = place.coordinate.longitude

    var zoom:Float = 17
    let addressesForZoom: [String] = place.formattedAddress!.components(separatedBy: ",")
    switch addressesForZoom.count{
    case 1:
        zoom = 5
    case 2:
        zoom = 10
    case 3:
        zoom = 14
    case 4:
        zoom = 17
    default:
        zoom = 17
    }

    let camera = GMSCameraPosition.camera(withLatitude: lat, longitude: long, zoom: zoom)
    mapView.camera = camera

    searchTextField.text = place.name
    commentsVC.nameText = place.name
    commentsVC.addressText = place.formattedAddress ?? ""
    MainViewController.placeId = place.placeID
    MainViewController.placeName = place.name
    MainViewController.placeAddress = place.formattedAddress ?? ""
    commentsVC.getComments()

    mapView.clear()
    let marker = GMSMarker()
    marker.position = CLLocationCoordinate2D(latitude: lat, longitude: long)
    marker.title = place.name
    marker.snippet = place.formattedAddress
    marker.map = mapView

    self.dismiss(animated: true, completion: nil)
}
```

e. CommentsViewController is nested by MainViewControler. It can be pulled up or drag down to maximum or minimum it’ s showing view

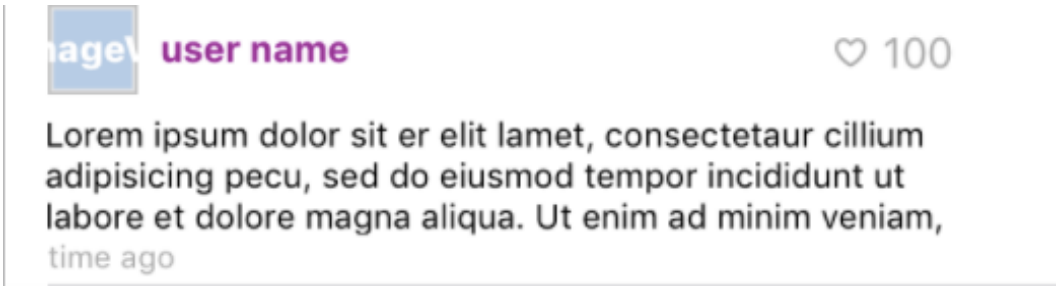




```swift
func addCommentsVC(atOffset offset:CGFloat) -> UIView?{
    let frameForView = self.view.bounds.offsetBy(dx: 0, dy: self.view.bounds.height - offset)

    let sb = UIStoryboard(name: "Main", bundle: nil)
    commentsVC = sb.instantiateViewController(withIdentifier: "commentsVC") as! CommentsViewControlle

    if let view = commentsVC.view{
        view.frame = frameForView
        view.layer.cornerRadius = 5
        view.layer.shadowOffset = CGSize(width: 2, height: 2)
        view.layer.shadowColor = UIColor.black.cgColor
        view.layer.shadowRadius = 3
        view.layer.shadowOpacity = 0.5

        self.addChild(commentsVC)
        self.view.addSubview(view)
        commentsVC.didMove(toParent: self)

        let panGestureRecongizer = UIPanGestureRecognizer(target: self, action:
            #selector(MainViewController.handlePan(gestureRecognizer:)))
        view.addGestureRecognizer(panGestureRecongizer)

        let collision = UICollisionBehavior(items: [view])
        collision.collisionDelegate = self
        animator.addBehavior(collision)

        let boundary = view.frame.origin.y+view.frame.size.height
        var boundaryStart = CGPoint(x: 0, y: boundary)
        var boundaryEnd = CGPoint(x:self.view.bounds.width, y:boundary)
        collision.addBoundary(withIdentifier: intID1, from: boundaryStart, to: boundaryEnd)

        boundaryStart = CGPoint(x:0, y:0)
        boundaryEnd = CGPoint(x: self.view.bounds.width, y: 0)
        collision.addBoundary(withIdentifier: intID2 , from: boundaryStart, to: boundaryEnd)

        gravity.addItem(view)

        let itemBehavior = UIDynamicItemBehavior(items: [view])
        animator.addBehavior(itemBehavior)
        return view
    }


@objc func handlePan(gestureRecognizer:UIPanGestureRecognizer){
    let touchPoint = gestureRecognizer.location(in: self.view)
    let draggedView = gestureRecognizer.view!

    if gestureRecognizer.state == .began{
        let dragStartPoint = gestureRecognizer.location(in: draggedView)

        if dragStartPoint.y<80 {
            viewDragging = true
            previousTouchPoint = touchPoint
        }
    }else if gestureRecognizer.state == .changed && viewDragging{
        let yOffset = previousTouchPoint.y - touchPoint.y
        draggedView.center = CGPoint(x: draggedView.center.x, y: draggedView.center.y-yOffset)
        previousTouchPoint = touchPoint
    }else if gestureRecognizer.state == .ended && viewDragging{
        //pin
        pin(view: draggedView)
        // addVelocity
        addVelocity(toView: draggedView, fromGestureRecognizer: gestureRecognizer)
        animator.updateItem(usingCurrentState: draggedView)
        viewDragging = false
    }

}
```

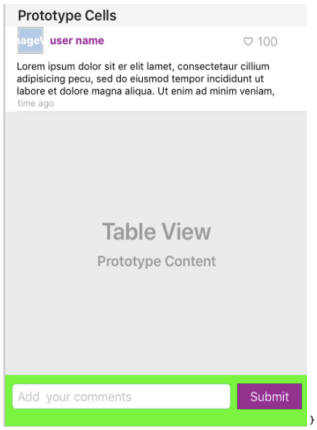
```
func pin(view:UIView){
    let viewHasReachedPinLocation = view.frame.origin.y < 180

    if viewHasReachedPinLocation{
        if !viewPinned{
            var snapPosition = self.view.center
            snapPosition.y += 70
            snap = UISnapBehavior(item: view, snapTo: snapPosition)
            animator.addBehavior(snap)
            viewPinned = true
        }
    }else{
        if viewPinned {
            animator.removeBehavior(snap)
            //setVisibility(view: view, alpha: 1)
            viewPinned = false
        }
    }
}

func addVelocity (toView view:UIView, fromGestureRecognizer panGesture:UIPanGestureRecognizer){
    var velocity = panGesture.velocity(in: self.view)
    velocity.x = 0

    if let behavior = itemBehavior(forView: view){
        behavior.addLinearVelocity(velocity, for: view)
    }
}


func itemBehavior (forView view:UIView) -> UIDynamicItemBehavior?{
    for behavior in animator.behaviors{
        if let itemBehavior = behavior as? UIDynamicItemBehavior{
            if let possibleView = itemBehavior.items.first as? UIView, possibleView == view {
                return itemBehavior
            }
        }
    }
    return nil
}

func collisionBehavior(_ behavior: UICollisionBehavior, beganContactFor item: UIDynamicItem, withBoundaryIdentifier identifier:
    NSCopying?, at p: CGPoint) {
    if identifier === intID2 {
        let view = item as! UIView
        pin(view:view)
    }
}
```

f. CommentsViewController shows all comments to this place. Each comment includes user photo, user name, number of likes, content, and duration time to now.



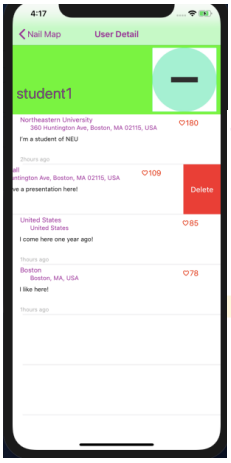
```swift
func timeAgo() -> String{
    let secondsAgo = Int64(Date().timeIntervalSince(self))

    let min:Int64 = 60
    let hour:Int64 = 60 * min
    let day:Int64 = 24 * hour
    let week:Int64 = 7 * day
    let month:Int64 = 30 * day
    let year:Int64 = 365 * day

    if secondsAgo < min {
        return "less than 1 min"
    }else if secondsAgo < hour {
        return "\(secondsAgo/min)mins ago"
    }else if secondsAgo < day {
        return "\(secondsAgo/hour)hours ago"
    }else if secondsAgo < week {
        return "\(secondsAgo/day)days ago"
    }else if secondsAgo < month {
        return "\(secondsAgo/week)weeks age"
    }else if secondsAgo < year {
        return "\(secondsAgo/month)months ago"
    }else {
        return "\(secondsAgo/year)years ago"
    }
}
```

g. CommentsViewController’ s buttom is where the user can write their own comments to this place and upload it to server.




```swift
func upLoadComment (username:String, userPhoto:[UInt8], placeId:String, placeName:String, placeAddress:String, content:String,
    publishDate:String, likes: Int){
    let jsonUrlStr = LoginViewController.serverUrl+"/stickies"
    guard let jsonUrl = URL(string: jsonUrlStr) else {return}
    var request = URLRequest(url:jsonUrl)
    request.httpMethod = "POST"
    request.setValue("application/json", forHTTPHeaderField: "Content-Type")
    let postJson: [String: Any] = ["username": username, "userPhoto":userPhoto, "placeId":placeId, "content":content,
        "placeName":placeName, "placeAddress":placeAddress, "publishDate":publishDate, "likes": likes]
    let postData = try? JSONSerialization.data(withJSONObject: postJson)
    //print(postData!)
    request.httpBody = postData

    URLSession.shared.dataTask(with: request){ (data, response, err) in
        //let httpResponse = response as? HTTPURLResponse
        //print(httpResponse!.statusCode)
        guard let data = data else {return}
        do{
            let comment = try JSONDecoder().decode(Comment.self, from: data)
            print(comment.content)
            DispatchQueue.main.async {
                CommentsViewController.comments.append(comment)
                self.tableView.reloadData()
                self.contentTextField.text = ""
            }
            //print(user.password)
            //print(user.photo)
        }catch let jsonErr{
            print("Error serializing json:", jsonErr)
        }
        }.resume()
}
```

h. When click user photo on the MainViewController’ s navigation right item, it will go to UserDetailViewController. It shows all comments the user post and can delete it.




```swift
func deleteComment(_ id:String){
    let jsonUrlStr = LoginViewController.serverUrl+"/stickies/"+id
    guard let jsonUrl = URL(string: jsonUrlStr) else {return}
    var request = URLRequest(url:jsonUrl)
    request.httpMethod = "DELETE"
    URLSession.shared.dataTask(with: request){ (data, response, err) in
        //let httpResponse = response as? HTTPURLResponse
        //print(httpResponse!.statusCode)
        if let data = data {
            print("Delete successfully")
        }else{
            print("Delete failed")
        }
        }.resume()
}
```

5. Server build

a. MongoBD as server database

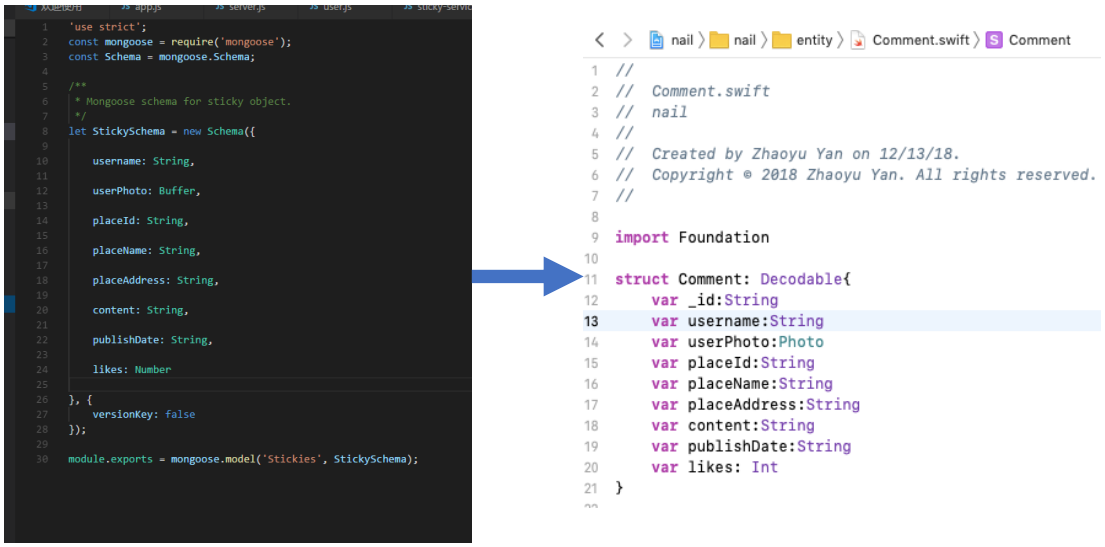
b. Structure of server: model -> service -> controller -> router

c. Model is same as entities in swift

```javascript
'use strict';
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

/**
 * Mongoose schema for sticky object.
 */
let UsersSchema = new Schema({

    username: String,

    password: String,

    photo: Buffer
}, {
    versionKey: false
});

module.exports = mongoose.model('Users', UsersSchema);
```

nail 〉 nail 〉 entity 〉 User.swift 〉 User

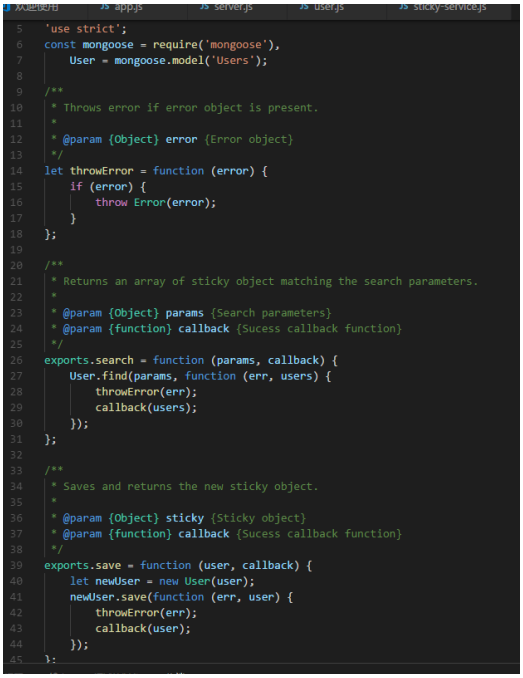
```swift
//
//  User.swift
//  nail
//
//  Created by Zhaoyu Yan on 12/11/18.
//  Copyright © 2018 Zhaoyu Yan. All rights reserved.
//

import Foundation

struct User: Decodable{
    var _id:String
    let username:String
    var password:String
    var photo: Photo

}
```



```
'use strict';
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

/**
 * Mongoose schema for sticky object.
 */
let StickySchema = new Schema({

    username: String,

    userPhoto: Buffer,

    placeId: String,

    placeName: String,

    placeAddress: String,

    content: String,

    publishDate: String,

    likes: Number

}, {
    versionKey: false
});

module.exports = mongoose.model('Stickies', StickySchema);
```

```
nail 〉 nail 〉 entity 〉 Comment.swift 〉 Comment

//
//  Comment.swift
//  nail
//
//  Created by Zhaoyu Yan on 12/13/18.
//  Copyright © 2018 Zhaoyu Yan. All rights reserved.
//

import Foundation

struct Comment: Decodable{
    var _id:String
    var username:String
    var userPhoto:Photo
    var placeId:String
    var placeName:String
    var placeAddress:String
    var content:String
    var publishDate:String
    var likes: Int
}
```

d. Service creates functions for MongoDB

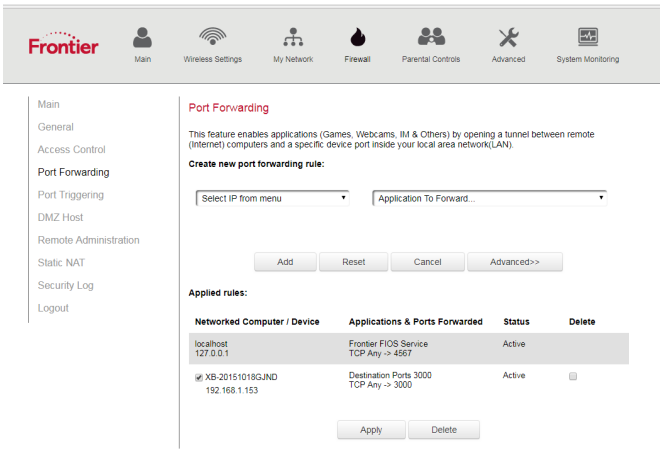


```
'use strict';
const mongoose = require('mongoose'),
    User = mongoose.model('Users');

/**
 * Throws error if error object is present.
 *
 * @param {Object} error {Error object}
 */
let throwError = function (error) {
    if (error) {
        throw Error(error);
    }
};

/**
 * Returns an array of sticky object matching the search parameters.
 *
 * @param {Object} params {Search parameters}
 * @param {function} callback {Sucess callback function}
 */
exports.search = function (params, callback) {
    User.find(params, function (err, users) {
        throwError(err);
        callback(users);
    });
};

/**
 * Saves and returns the new sticky object.
 *
 * @param {Object} sticky {Sticky object}
 * @param {function} callback {Sucess callback function}
 */
exports.save = function (user, callback) {
    let newUser = new User(user);
    newUser.save(function (err, user) {
        throwError(err);
        callback(user);
    });
};
```

e. Controller creates methods for web protocol

f. Router sets urls

g. Set port forwarding of my wifi router to public my server port



h. Access to server by url: http://24.62.61.206:3000/users & http://24.62.61.206:3000/stickies