



SDS Group 3: Hi Fi Prototype

Initial directions

How they want components stacked?

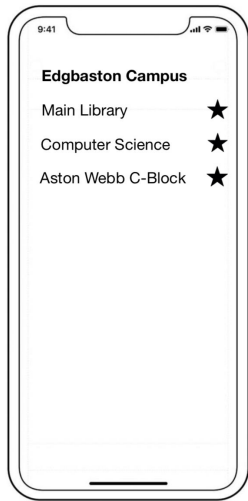
- Favorites first?
- configurable?
- How it affects ease of use
- Does it boost engagement
- windowed?

Navigation

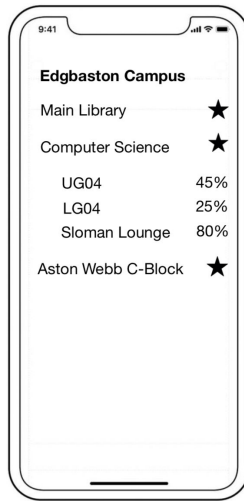
- burger menu?
 - Tree?
 - Drop down?
- } what do they want to hit?

User Interface

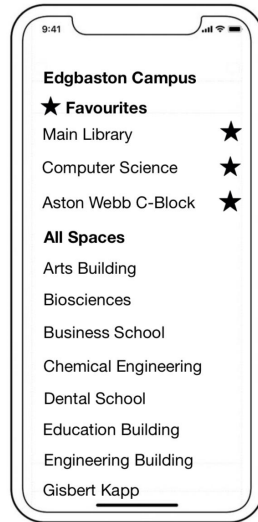
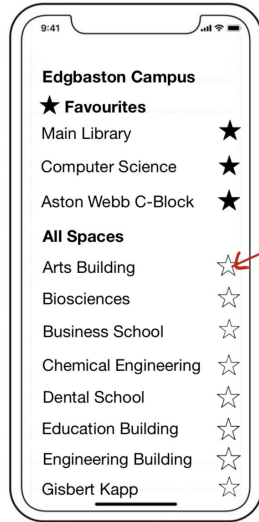


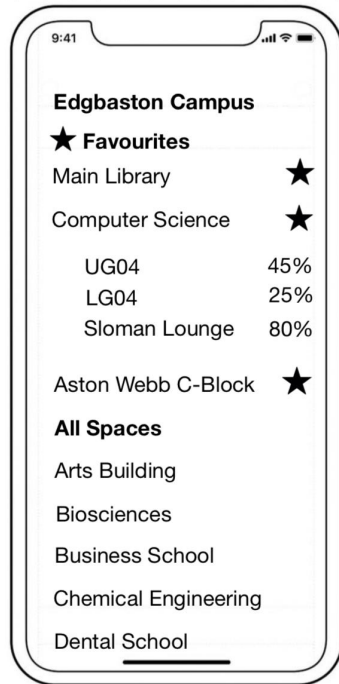


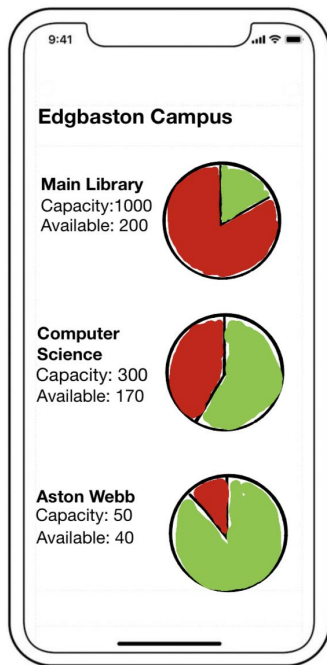
Favourites
Page

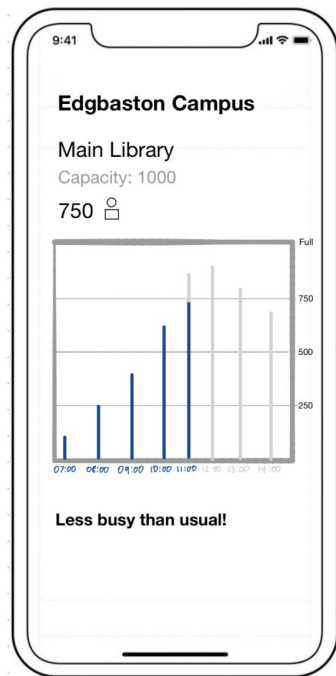


} Are percentages
a good
indicator?











HiFi Prototype: current status

Working / Implemented

- PIR interface with arduino
- Arduino interface with Raspberry PI over Serial
- Raspberry PI connection to InfluxDB
- InfluxDB connection to Go web API
- Go web API connection with frontend angular webapp

Remaining / requires further work

- Higher resolution data from PIR sensor, rather than binary output, ability to interpret direction of motion => person leaving or entering room.
- Logic between rooms/ zones, i.e. registration of person leaving RoomA via door to RoomB => roomA_count -- ; roomB_count ++

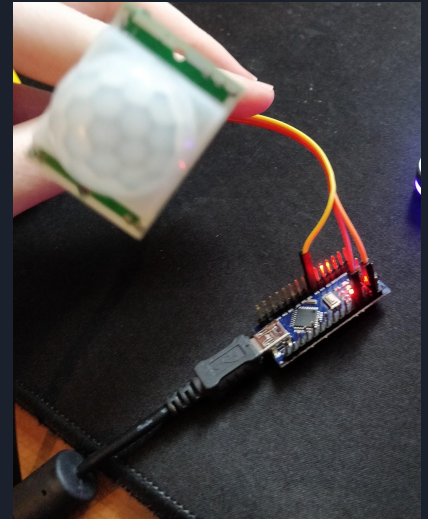
HiFi Prototype Component: Arduino

Using the headers soldered onto our Arduino Nano, we have connected a PIR sensor. The PIR sensor allows us to detect motion with a wide aperture and decent range.

Currently we are only using this sensor to produce a binary output as to whether motion has been detected. This data is transmitted over the serial connection to the raspberry PI running a simple go script

The PIR sensor has the capability to give us much more detailed information that we could use to interpret the direction of movement of a target, allowing us to register people leaving and entering a room from a single sensor.

A downside of this sensor is that it requires a *cool down* period between detections of around 6 seconds, reducing the accuracy of a deployed sensor in cases of rapid increase in occupancy.





HiFi Prototype Component: Arduino

To further improve upon this element of our design, we should look into using higher resolution data from the sensor as well as looking into alternative sensors for the outlined cases where this one fails.

The accuracy of the sensor is also questionable with it failing to register motion in some cases, this may be solved by further tweaks to the sensitivity via the potentiometers on the PCB


```
int ledPin = LED_BUILTIN;           // choose the pin for the LED
int inputPin = 2;                   // choose the input pin (for PIR sensor)
int pirState = LOW;                 // we start, assuming no motion detected
int val = 0;                        // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);          // declare LED as output
  pinMode(inputPin, INPUT);         // declare sensor as input
  digitalWrite(ledPin, LOW);
  Serial.begin(9600);
}

void loop(){
  val = digitalRead(inputPin); // read input value
  if (val == HIGH) // check if the input is HIGH
  {
    digitalWrite(ledPin, HIGH); // turn LED ON

    if (pirState == LOW)
    {
      Serial.println("1"); // print on output change
      pirState = HIGH;
    }
  }
  else
  {
    digitalWrite(ledPin, LOW); // turn LED OFF

    if (pirState == HIGH)
    {
      Serial.println("0"); // print on output change
      pirState = LOW;
    }
  }
}
```



HiFi Prototype Component: Sensor Reading Relay

This component acts as an intermediary between the sensor readings from the arduino and the Real-time Influx Database.

It takes the form of a script written in golang which runs on the raspberry PI connected to the Arduino via a serial cable.

Initially, upon running this script, a connection is made to the database using a token that is injected a build time along with a unique RoomID for the room the hub is placed in. The script then initialises a occupancy counter by fetching the most recent reading from the database.

Upon receiving data over the serial connection the script determines if it is a valid reading, currently only accepting values of “1” to signify motion detected => person entering the room. If valid, the script increments its local counter and writes the updated value back to the database.

```
ubuntu@ubuntu:~/ROT$ sudo systemctl status pi-db.service
● pi-db.service - "Influx DB sensor logger"
   Loaded: loaded (/etc/systemd/system/pi-db.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2020-02-28 15:25:58 UTC; 4s ago
     Main PID: 1928 (pi)
        Tasks: 9 (limit: 1039)
      CGroup: /system.slice/pi-db.service
              └─1928 /home/ubuntu/ROT/bin/pi

Feb 28 15:25:58 ubuntu systemd[1]: Started "Influx DB sensor logger".
Feb 28 15:25:58 ubuntu pi[1928]: Setup for room: 1001
Feb 28 15:25:58 ubuntu pi[1928]: initialising room counter
Feb 28 15:25:59 ubuntu pi[1928]: 2020/02/28 15:25:59 multiple rooms sharing and ID
Feb 28 15:25:59 ubuntu pi[1928]: Initial Occupancy: 3.000000Setup for Building: 0, floor: 0
Feb 28 15:25:59 ubuntu pi[1928]: 2020/02/28 15:25:59 Searching for arduino
Feb 28 15:25:59 ubuntu pi[1928]: 2020/02/28 15:25:59 ttyUSB0
ubuntu@ubuntu:~/ROT$
```

HiFi Prototype Component: Sensor Reading Relay: Code

```
package main

import (
    "context"
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "net/http"
    "strconv"
    "strings"
    "time"

    "github.com/huin/goserial" // You, 16 hours ago · working arduino → website
    "github.com/influxdata/influxdb-client-go"
)
```

```
func findArduino() string {
    log.Println("Searching for arduino")
    contents, err := ioutil.ReadDir("/dev/")
    if err != nil {
        log.Fatal(err)
    }
    for _, f := range contents {
        if strings.Contains(f.Name()), "tty.usbserial" || strings.Contains(f.Name(), "ttyUSB") {
            log.Println(f.Name())
            return "/dev/" + f.Name()
        }
    }
    return ""
}

func readSerial(port io.ReadWriteCloser) {
    buf := make([]byte, 128)
    n, _ := port.Read(buf)
    if n != 0 {
        log.Printf("%q\n", buf[:n])
    }
    readSerial(port)
}
```

// SensorData is a Struct for storing information resulting from InfluxDB Query

```
You, 16 hours ago | 1 author (You)
type SensorData struct {
    Time      time.Time "flux:"_time" json:"time"
    Field     string    "flux:"_field" json:"field"
    Value     float64   "flux:"_value" json:"value"
    BuildingID int       "flux:"_BuildingID" json:"BuildingID"
    FloorID   int       "flux:"_FloorID" json:"FloorID"
    RoomID    int       "flux:"_RoomID" json:"RoomID"
}
```

```
// DBConnect given an access token creates an influxdb client for interfacing with the DB
func DBConnect(InfluxToken string) (*influxdb.Client, error) {
    // You can generate a token from the 'Tokens Tab' in the UI
    client := http.Client{
        return influxdb.New("https://us-central1-gcp.cloud2.influxdata.com", InfluxToken, influxdb.WithHTTPClient(&client))
    }
}

func initRoomCounter(db *influxdb.Client) (int, error) {
    println("initialising room counter")
    q := fmt.Sprintf("from(bucket: \"my-test-bucket\")
    D range(start: %s)
    D filter(fn: (r) => r.RoomID == \"%s\")
    D last(), \"-38d\", RoomID)

    resp, err := db.QueryCSV(context.Background(), q, "833c7fbc1d19c9be")
    if err != nil {
        return 0, err
    }
    readings := make([]SensorData, 0)
    for resp.Next() {
        reading := SensorData{}
        err = resp.Unmarshal(&reading)
        if err != nil {
            return 0, err
        }
        readings = append(readings, reading)
    }
    if len(readings) == 0 {
        log.Println("No preexisting data for room")
        return 0, nil
    } else if len(readings) > 1 {
        log.Println("multiple rooms sharing and ID")
        BuildingID := strconv.Itoa(readings[0].BuildingID)
        fmt.Println("Initial Occupancy: %f", readings[0].Value)
        FloorID := strconv.Itoa(readings[0].FloorID)
        return int(readings[0].Value), nil
    } else {
        BuildingID := strconv.Itoa(readings[0].BuildingID)
        FloorID := strconv.Itoa(readings[0].FloorID)
        fmt.Println("Initial Occupancy: %f", readings[0].Value)
        return int(readings[0].Value), nil
    }
}
```

```
func updateRoomCounter(db *influxdb.Client, occupancy int) error {
    myMetric := []influxdb.Metric{
        influxdb.NewMetric(
            map[string]interface{}{"occupancy": occupancy},
            "Sensor Readings",
            map[string]string{"BuildingID": BuildingID, "FloorID": FloorID, "RoomID": RoomID},
            time.Now(),
        )
    }
    _, err := db.Write(context.Background(), "my-test-bucket", "833c7fbc1d19c9be", myMetric...)
    return err
}
```

```
var (
    // InfluxToken allows access to the InfluxDB database BUILDTIME INJECTION
    InfluxToken string
    // RoomID stores the roomID of the room the system running this script is deployed to BUILDTIME INJECTION
    RoomID string
    // FloorID stores the floorID of the room specified by RoomID
    FloorID string
    // BuildingID stores the buildingID of the room specified by RoomID
    BuildingID string
)

func main() {
    fmt.Printf("Setup for room: %s\n", RoomID)
    println(InfluxToken)
    db, err := DBConnect(InfluxToken)
    if err != nil {
        log.Println(err)
    }
    roomCounter, err := initRoomCounter(db)
    fmt.Printf("Setup for Building: %s, floor: %s\n", BuildingID, FloorID)
    if err != nil {
        log.Println(err)
    }
    c := goserial.Config{Name: findArduino(), Baud: 9600}
    s, err := goserial.OpenPort(c)
    if err != nil {
        log.Fatal(err)
    }
    for {
        buf := make([]byte, 128)
        n, _ := s.Read(buf)
        sval := string(buf[:n])
        ival, err := strconv.Atoi(strings.ReplaceAll(strings.ReplaceAll(sval, "\n", ""), "\r", ""))

        if err != nil {
            log.Println(err)
        }
        log.Println(ival)
        if !(ival > 1) {
            roomCounter += ival
        }
        err = updateDBRoomCounter(db, roomCounter)
        if err != nil {
            log.Println(err)
        }
    }
}
```

You, 16 hours ago · working arduino → website

HiFi Prototype Component: InfluxDB

The backbone of any data-driven application is its database. For our application we have trialled using a time series database, specifically InfluxDB. This allows us to store data and display data as it is gathered, in real time.

Alternative solutions would be Google Firebase and its real time database implementation. This would arguably be a better fit for our application as we are already using firebase to host other sections of the service.

InfluxDB provides us with an intuitive web interface for visualising raw data as well as *bucket-ised* data collection with tags enabling us to filter data by room, floor or building.

Another downside of InfluxDB is that each bucket has a maximum retention of 30 days, severely limiting the amount of historical data we can gather for use in machine learning applications.



HiFi Prototype Component: Web API

- As an intermediary between the database and our frontend web-app we have a simple web API written in Golang. This has an endpoint `/occupancy/{roomId|buildingID|floorID}`
- Using this, upon loading a page that requires room occupancy information, a request is made based on the whether the occupancy of a room, building or floor is required.
- If the occupancy of the floor or building is requested the API simply returns the sum of the individual room occupancies of that given zone.
- By having this API we reduce client-side load and secure access to our DB behind a compiled server-side binary.
-

```
λ make server
make[1]: Entering directory '/home/sam/git-repos/ROT'
SUCCESS Compiled server binary
make[1]: Leaving directory '/home/sam/git-repos/ROT'
Running server
2020/02/28 15:57:28 Calculating Occupancy
from(bucket: "my-test-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r.RoomID == "1001")
  |> last()
2020/02/28 15:57:28 Executed query
READING
{29}
```

Web API

```
λ curl 'localhost:6969/occupancy/?roomId=1001'
{"occupancy":29}
```

Client

HiFi Prototype Component: Web API: Code

```
package main // You, 2 days ago · Merge to go webserver (R18)
```

```
import (
    "context"
    "encoding/json"
    "fmt"
    "html"
    "log"
    "net/http"
    "time"

    "github.com/gorilla/mux"
    "github.com/influxdata/influxdb-client-go"
)

// DB encapsulates the database connection so that it might be shared between response handlers
type DB struct {
    Client *influxdb.Client
}

// InfluxToken is injected at compile-time and authorizes access to the Influx Database
var InfluxToken string

// NewDB creates a DB struct which abstracts the InfluxDB connection.
// Allowing for http response methods to share the same connection
func NewDB(token string) DB {
    client := http.Client{}
    iDBClient, err := influxdb.New("https://us-central-1-gcp.cloud2.influxdata.com", token, influxdb.WithHTTPClient(&client))
    if err != nil {
        log.Fatalf(err)
    }
    return DB{Client: iDBClient}
}

func main() {
    db := NewDB(InfluxToken)
    router := mux.NewRouter().StrictSlash(true)

    router.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "You have reached the toplevel of the RoI-IoI internal web server, the following endpoints are available: %q", html.EscapeString(r.URL.Path))
    })

    router.HandleFunc("/occupancy/", db.calcOccupancy)

    log.Fatal(http.ListenAndServe(":1999", router))
}

// SensorData allows for important information from InfluxDB query responses to be accessed structurally
type SensorData struct {
    Time time.Time `flux:"time" json:"time"`
    Field string `flux:"field" json:"field"`
    Value float64 `flux:"value" json:"value"`
}
```

```
type Response struct {
    Occupancy float64 `json:"occupancy"`
}

// ErrorResponse allows for errors to be sent by the API on bad requests
type ErrorResponse struct {
    Response string `json:"response"`
}

func (db *DB) calcOccupancy(w http.ResponseWriter, r *http.Request) {
    log.Println("Calculating Occupancy")
    vars := r.URL.Query()
    var (
        buildingID string
        floorID string
        roomID string
    )
    buildingIDs := vars["buildingID"]
    if len(buildingIDs) != 0 {
        buildingID = buildingIDs[0]
    } else {
        buildingID = ""
    }
    floorIDs := vars["floorID"]
    if len(floorIDs) != 0 {
        floorID = floorIDs[0]
    } else {
        floorID = ""
    }
    roomIDs := vars["roomID"]
    if len(roomIDs) != 0 {
        roomID = roomIDs[0]
    } else {
        roomID = ""
    }
    var q string
    if buildingID != "" {
        q = fmt.Sprintf("from(bucket: \"my-test-bucket\")\n" +
            "  range(start: %s)\n" +
            "    filter(fn: (r) => r.BuildingID == \"%s\")\n" +
            "    last(), \"-30d\", buildingID)\n" +
            "  } else if floorID != \"\" {\n" +
            "    q = fmt.Sprintf(\"from(bucket: \"my-test-bucket\")\n" +
            "      range(start: %s)\n" +
            "      filter(fn: (r) => r.FloorID == \"%s\")\n" +
            "      last(), \"-30d\", floorID)\n" +
            "    } else if roomID != \"\" {\n" +
            "      q = fmt.Sprintf(\"from(bucket: \"my-test-bucket\")\n" +
            "        range(start: %s)\n" +
            "        filter(fn: (r) => r.RoomID == \"%s\")\n" +
            "        last(), \"-30d\", roomID)\n" +
            "      }\n" +
            "    %s", time.Now().Format(time.RFC3339), buildingID, floorID, roomID, q)
    } else {
        q = fmt.Sprintf("from(bucket: \"my-test-bucket\")\n" +
            "  range(start: %s)\n" +
            "    last(), \"-30d\", roomID)\n" +
            "  }\n" +
            "  %s", time.Now().Format(time.RFC3339), q)
    }
    q = fmt.Sprintf("select mean(value) as occupancy from %s", q)
    query := influxdb.NewQuery(q)
    ctx := context.Background()
    readings, err := db.Client.QueryCSV(ctx, query, influxdb.DefaultOptions)
    if err != nil {
        log.Fatalf(err)
    }
    response := Response{Occupancy: 0}
    for resp := range readings {
        response.Occupancy += resp.Value
    }
    w.WriteHeader(http.StatusOK)
    json.NewEncoder(w).Encode(response)
}
```

```
} else {
    response := ErrorResponse{Response: "must provide a building, floor or room ID"}
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusCreated)
    json.NewEncoder(w).Encode(response)
    return
}

println(q)
resp, err := db.Client.QueryCSV(context.Background(), q, "833c7fbc1d19c9be")
if err != nil {
    log.Fatalf(err)
}
log.Println("Executed query")
var response Response
readings := make([]SensorData, 0)
for resp.Next() {
    println("READING")
    reading := SensorData{}
    err = resp.Unmarshal(&reading)
    if err != nil {
        log.Fatalf(err)
    }
    readings = append(readings, reading)
}

if len(readings) == 0 {
    println("ERROR: no data in database") // todo check if this is desired behaviour
} else if len(readings) > 1 {
    sumOcc := 0.0
    for _, each := range readings {
        sumOcc += each.Value
        response = Response{Occupancy: sumOcc}
    }
} else {
    response = Response{Occupancy: readings[0].Value}
}

fmt.Printf("%v\n", response)
w.Header().Set("Content-Type", "application/json")
w.WriteHeader(http.StatusOK)
json.NewEncoder(w).Encode(response)
```




HiFi Prototype Component: Frontend Web-App

- Finally, we have the user-facing frontend web-app
- Here we display the occupancy figures for each room on the system.

3: V[Alacritty Navigator]

motiondetect.ino - ROT - Visual Studio Code

```
> range(start: -30d)
> filter(fn: (r) => r.BuildingID == "1")
> last()
```

2020/02/28 15:55:18 Calculating Occupancy from(bucket: "my-test-bucket")

```
> range(start: -30d)
> filter(fn: (r) => r.BuildingID == "2")
> last()
```

2020/02/28 15:55:18 Calculating Occupancy from(bucket: "my-test-bucket")

```
> range(start: -30d)
> filter(fn: (r) => r.BuildingID == "3")
> last()
```

2020/02/28 15:55:19 Executed query

READING

{29}

2020/02/28 15:55:19 Executed query

ERROR: no data in database

ERROR: no data in database

{0}

2020/02/28 15:55:19 Executed query

ERROR: no data in database

{0}

{0}

2020/02/28 15:55:19 Executed query

ERROR: no data in database

{0}

{0}

{0}

```
λ make site
```

```
cd ./web/rot && ng serve
```

10% building 3/3 modules 0 active[HPM] Proxy created: /api -> http://localhost:6969/

[HPM] Proxy rewrite rule created: "/api" -> ""

[HPM] Subscribed to http-proxy events: ['error', 'close']

chunk {main} main.js, main.js.map (main) 57.3 kB [initial] [rendered]

chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 140 kB [initial] [rendered]

chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]

chunk {scripts} scripts.js, scripts.js.map (scripts) 346 kB [entry] [rendered]

chunk {styles} styles.js, styles.js.map (styles) 9.71 kB [initial] [rendered]

chunk {vendor} vendor.js, vendor.js.map (vendor) 3.09 MB [initial] [rendered]

Date: 2020-02-28T15:54:59.376Z · Hash: 3142ecb47a542cd62eb8 · Time: 5933ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

: Compiled successfully.

[HPM] Rewriting path from "/api/occupancy/?roomId=1001" to "/occupancy/?roomId=1001"

[HPM] GET /api/occupancy/?roomId=1001 -> http://localhost:6969/

[HPM] Rewriting path from "/api/occupancy/?roomId=1002" to "/occupancy/?roomId=1002"

[HPM] GET /api/occupancy/?roomId=1002 -> http://localhost:6969/

[HPM] Rewriting path from "/api/occupancy/?roomId=1003" to "/occupancy/?roomId=1003"

[HPM] GET /api/occupancy/?roomId=1003 -> http://localhost:6969/

[HPM] Rewriting path from "/api/occupancy/?buildingID=1" to "/occupancy/?buildingID=1"

[HPM] GET /api/occupancy/?buildingID=1 -> http://localhost:6969/

[HPM] Rewriting path from "/api/occupancy/?buildingID=2" to "/occupancy/?buildingID=2"

[HPM] GET /api/occupancy/?buildingID=2 -> http://localhost:6969/

[HPM] Rewriting path from "/api/occupancy/?buildingID=3" to "/occupancy/?buildingID=3"

[HPM] GET /api/occupancy/?buildingID=3 -> http://localhost:6969/

□

~/git-repos/ROT w/Angular +1 +1

0:ssh-1:make*

A Rot

localhost:4200

ROT

Link Dropdown ▾

Log In [Sign Up](#)

Top Places

Sensor Test Room

200%

Room 103

0%

Room 102

0%

Top Locations

Guild of Students

0%

Library

0%

CS

0%