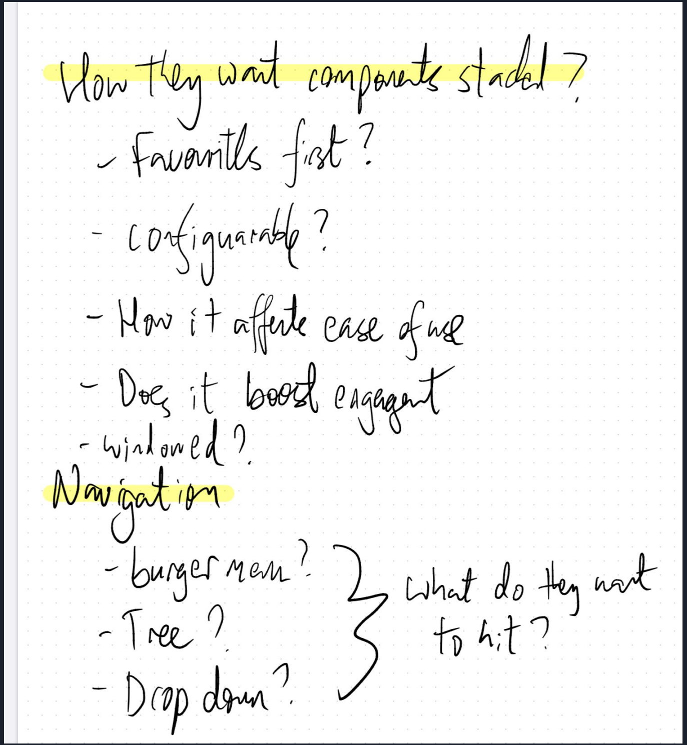
SDS Group 3: Hi Fi Prototype

Initial directions



How they want components stacked?

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

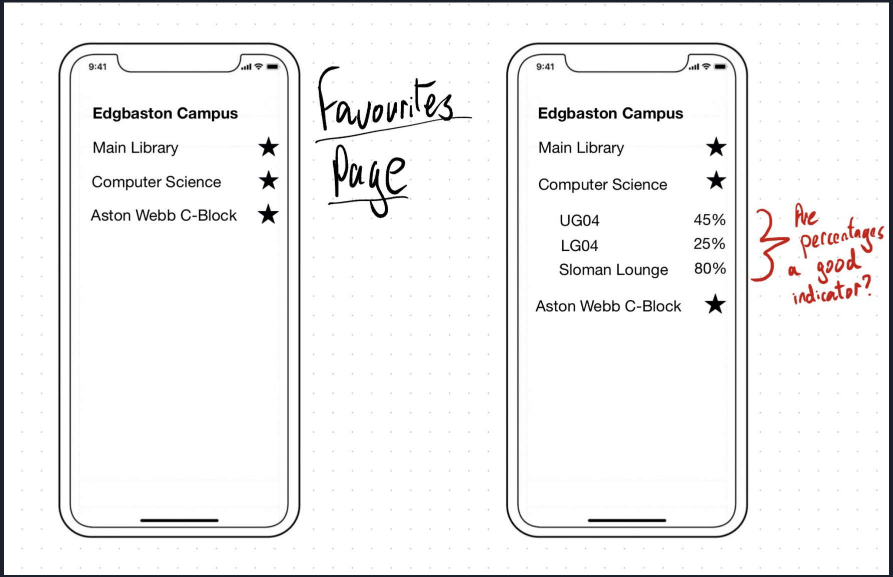
Navigation

- burger menu?
- Tree?
- Drop down?

} what do they want to hit?

Mobile Interface

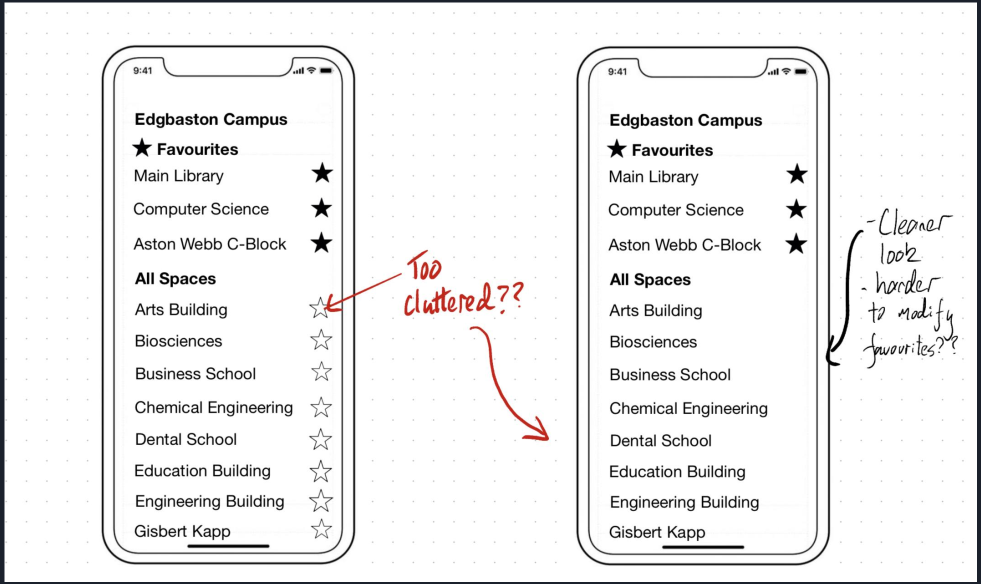
Favourites Page Initial Mock Up



The favourites page was designed to be the location where the majority of the time would be spent on the app.

The thoughts of this were that the user would be able to quickly check their preferred locations to study on campus and see exactly how close they were to capacity.

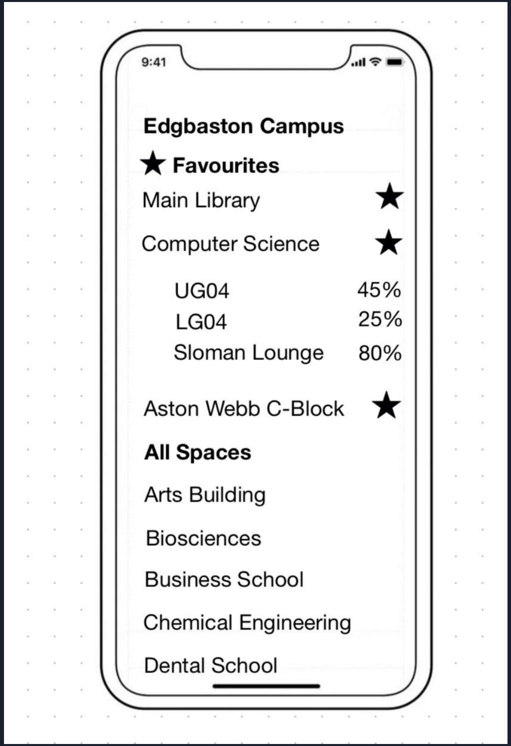
Favourites page - Extended



The next iteration of the favourites page resulted in showing all of the available spaces on campus with the favourites prioritised at the top of the page.

We experimented with the ability to quickly add new locations to the favourites with a toggle star, however this was seen to clutter the screen with a lot of unnecessary buttons.

Favourites Page - Version 1


9:41

Edgbaston Campus

★ **Favourites**

Main Library	★
Computer Science	★
UG04	45%
LG04	25%
Sloman Lounge	80%
Aston Webb C-Block	★

All Spaces

Arts Building

Biosciences

Business School

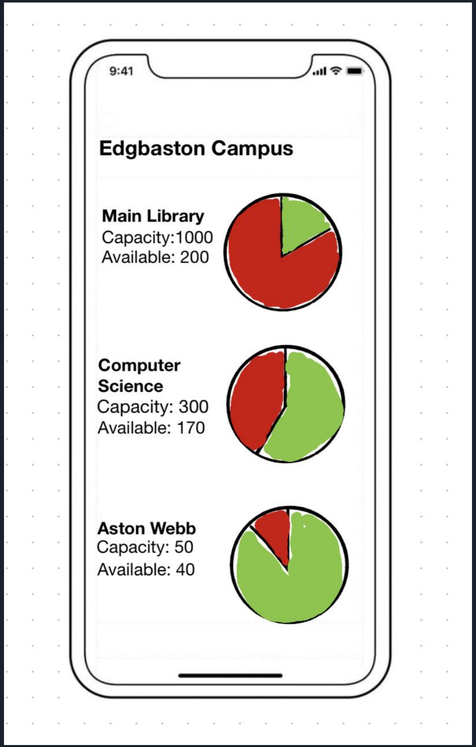
Chemical Engineering

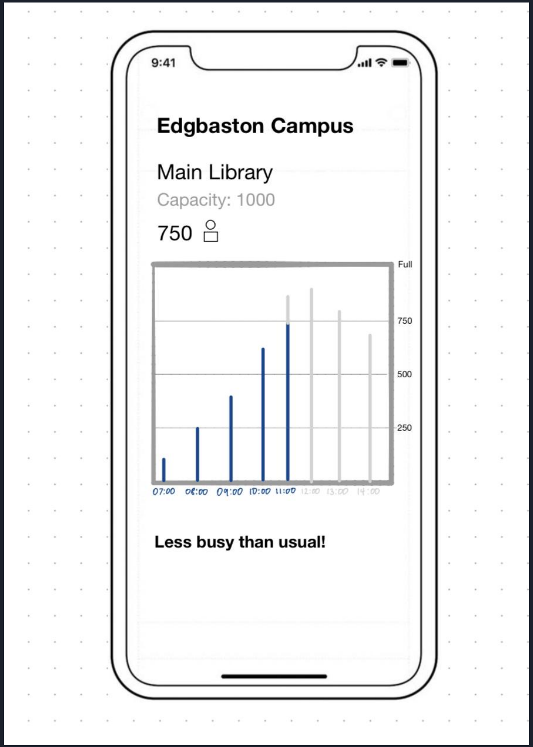
Dental School

The result of multiple iterations was our first main draft for the favourites menu.

Selecting building location would drop down all the available spaces within the building.

Data Representation

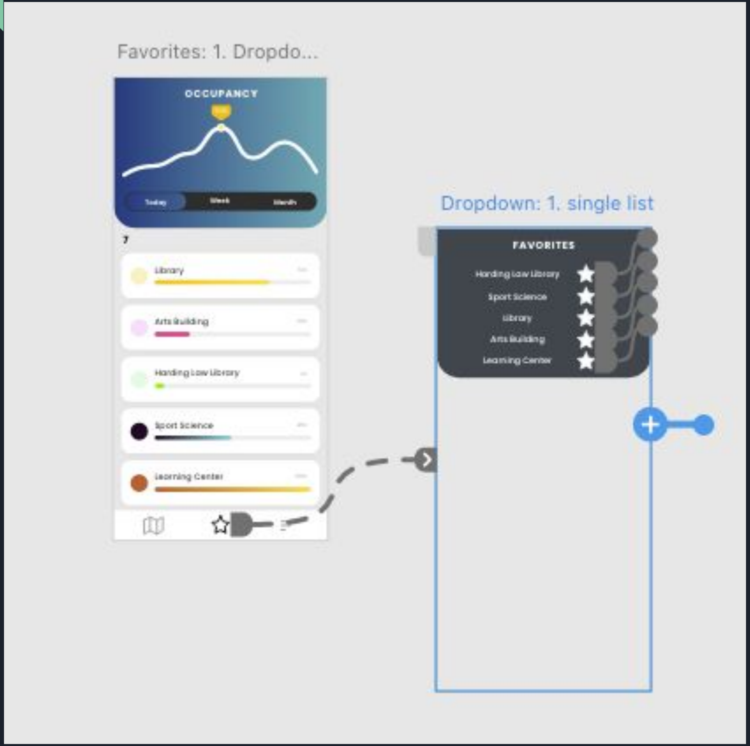




We iterated through multiple ways of representing the data. The first being pie charts on the favourites page to show the percentage free space in each building. However, it was thought the pie charts could be misleading in terms of actual free seats available.

The next iteration was to use bar charts. Using this, we would be able to show time series data of the closeness to capacity. The result being able to show how many people are predicted to be using the study space at certain times.

Mobile Landing Page

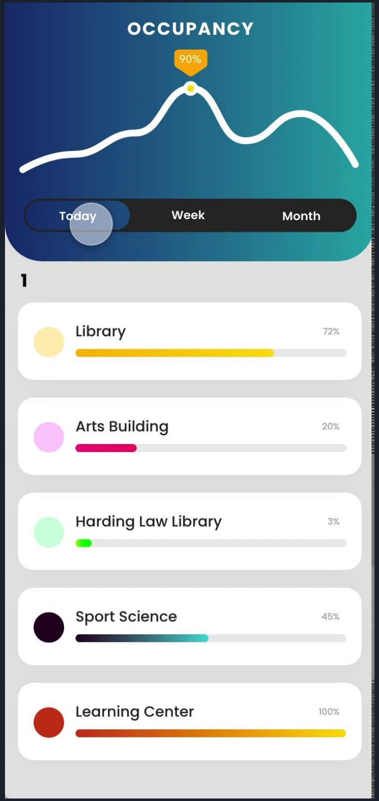


A series of interactive prototypes have been developed as to be presented to target users as to further refine designs.

The current state of prototypes can be found here (https://xd.adobe.com/view/098fc577-7bc2-4a30-6a89-8fdde5da82a5-7fbe/?fullscreen)

And contains the key iterations of the process.

Mobile Landing Page



Library 72%

Arts Building 20%

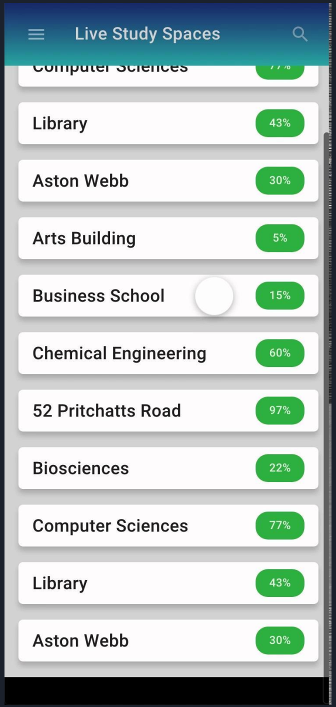
Harding Law Library 3%

Sport Science 45%

Learning Center 100%

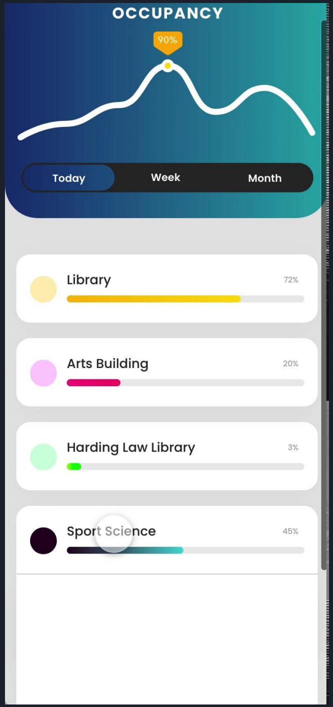
By dividing the display into two distinct sections, easily visualized data can be prominently displayed without detracting from the rest of the UI

Multi vs Single View



Live Study Spaces

Library	43%
Aston Webb	30%
Arts Building	5%
Business School	15%
Chemical Engineering	60%
52 Pritchatts Road	97%
Biosciences	22%
Computer Sciences	77%
Library	43%
Aston Webb	30%



OCCUPANCY

90%

Today Week Month

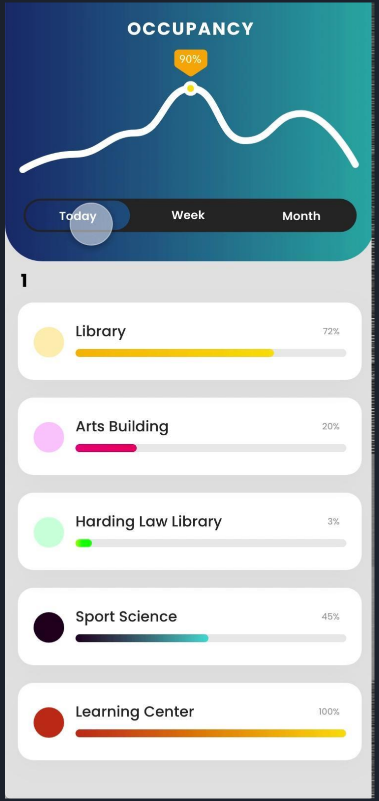
Library	72%
Arts Building	20%
Harding Law Library	3%
Sport Science	45%

Starting with these two initial concepts we started exploring these two different styles of designs.

Multiple views allows for much more segmentation of information and through links in a ‘Burger’ menu offer quick and easy access to specific pages.

Giving the perceived experience of staying on one single display make for a much more seamless experience as opposed to switching through screens.

Mobile UI


OCCUPANCY

90%

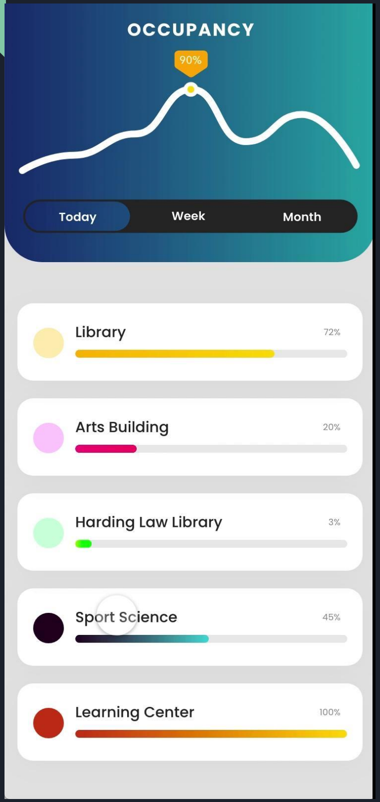
Today Week Month

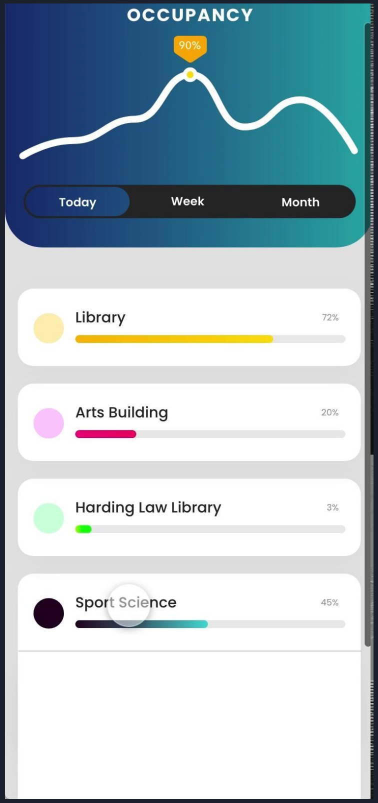
1

Library	72%
Arts Building	20%
Harding Law Library	3%
Sport Science	45%
Learning Center	100%

By dividing the display into two distinct sections, easily visualized data can be prominently displayed without detracting from the rest of the UI.

Mobile UI

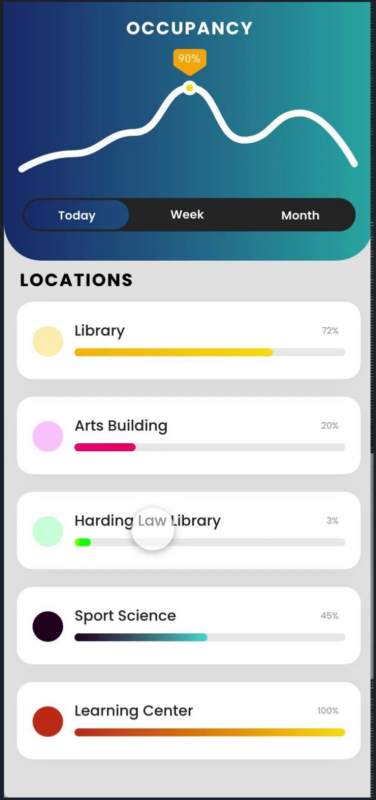




This is one of the first well received designs.

By focusing the user in on a single ‘card’ instead of allowing them to open up multiple cards simultaniuosly, it stops the whole screen being filled which detracts from the target information.

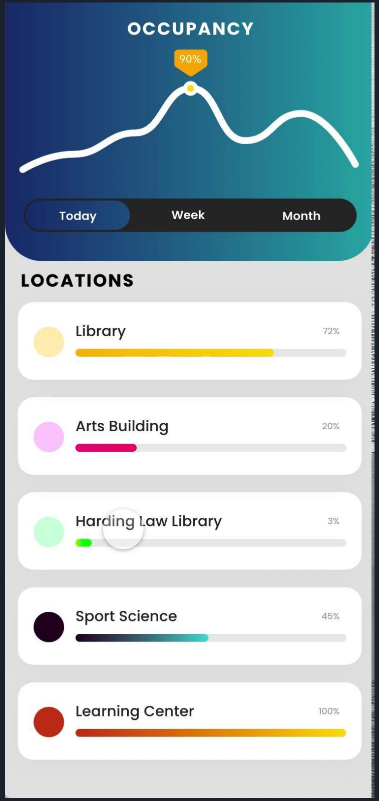
Mobile UI



By ‘pushing’ other cards out of the way it gives the appearance of more movement while minimising the amount of movement from the card selected, reducing the potential stress from repeatedly tracking objects across the screen.

The selected card is presented prominently by ‘fading out’ the pushed cards.

Mobile UI

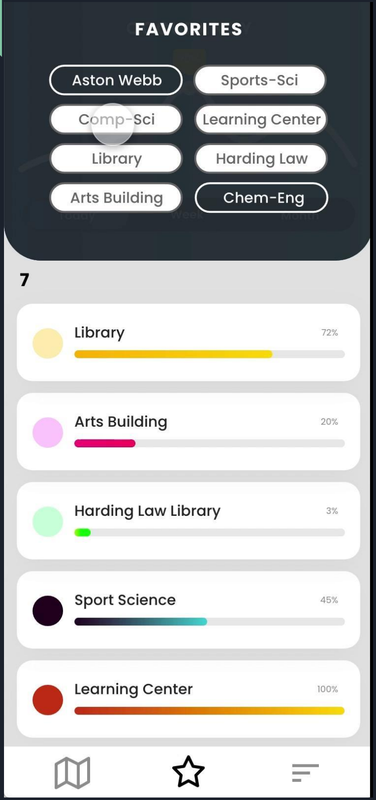


Similar to the previous design, cards can be expanded to take up a much more significant portion of the screen.

This has the added advantage of meaning more space for relevant information to be rendered.

Having the side effect of blocking out more of the screen, this could be taken as another step in the direction of a more focused and immersive view (Circling back towards a multi-screen style interaction).

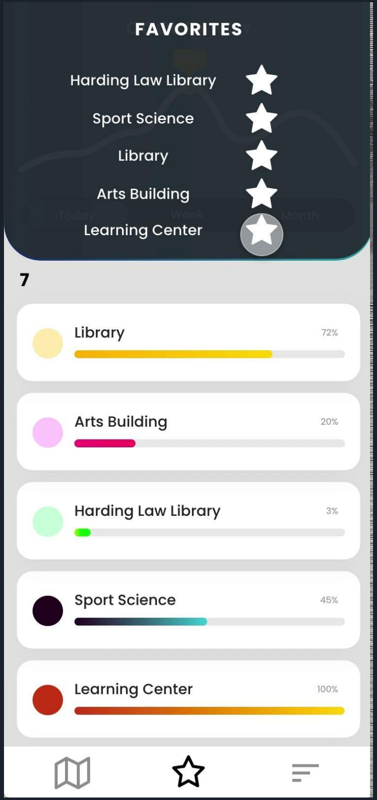
Mobile UI - Favorites


FAVORITES

Aston Webb
Sports-Sci
Comp-Sci
Learning Center
Library
Harding Law
Arts Building
Chem-Eng

7

Library 72%
Arts Building 20%
Harding Law Library 3%
Sport Science 45%
Learning Center 100%


FAVORITES

Harding Law Library
Sport Science
Library
Arts Building
Learning Center

7

Library 72%
Arts Building 20%
Harding Law Library 3%
Sport Science 45%
Learning Center 100%

One of the next components to consider is the previously mentioned ‘favorites’. In the end product allowing users to configure what they want displayed upon opening the application allows for simple customisation.

Mobile UI

The current prototype still needs to be put in front of more users to see if the current structure is sufficiently accepted along with what further functionality would actually be seen as useful.

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 ++

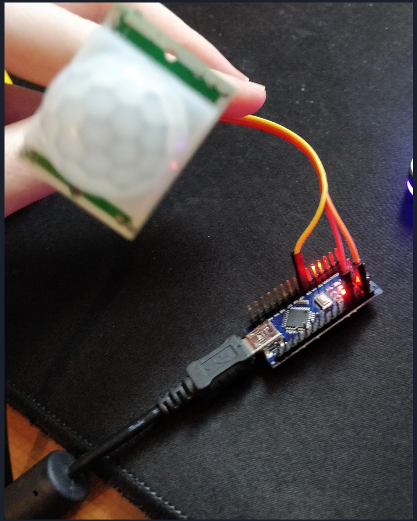
HiFi Prototype Component: Arduino

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.

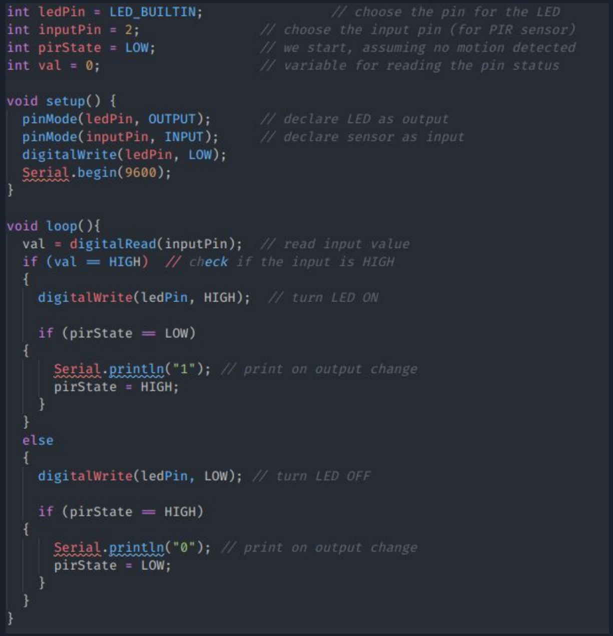
A downside of this sensor is that it requires a *cool down* period between detections of around 6 seconds, reducing the accuracy of a deployed sensor in cases of rapid increase in occupancy.



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;
    }
  }
}
```

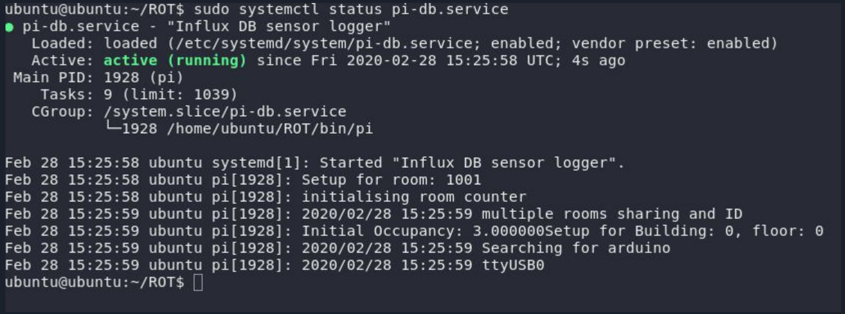
HiFi Prototype Component: Sensor Reading Relay

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.

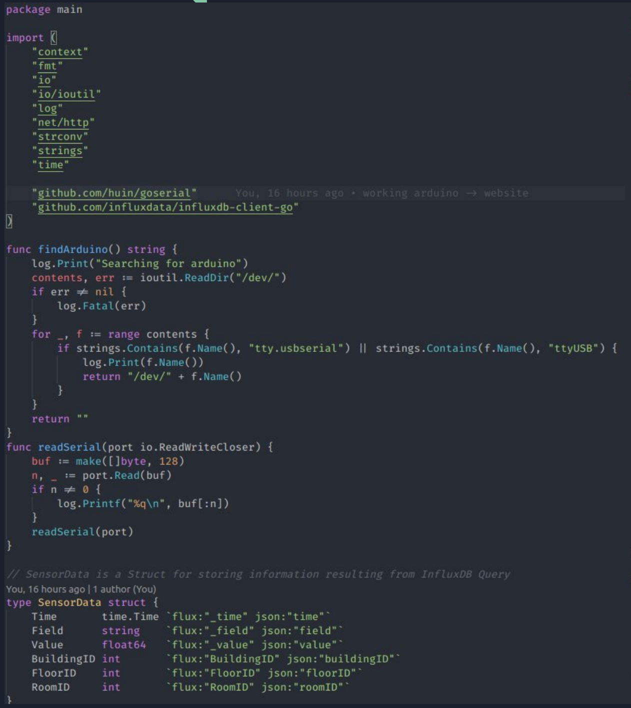
Initially, upon running this script, a connection is made to the database using a token that is injected a build time along with a unique RoomID for the room the hub is placed in. The script then initialises a occupancy counter by fetching the most recent reading from the database.

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.

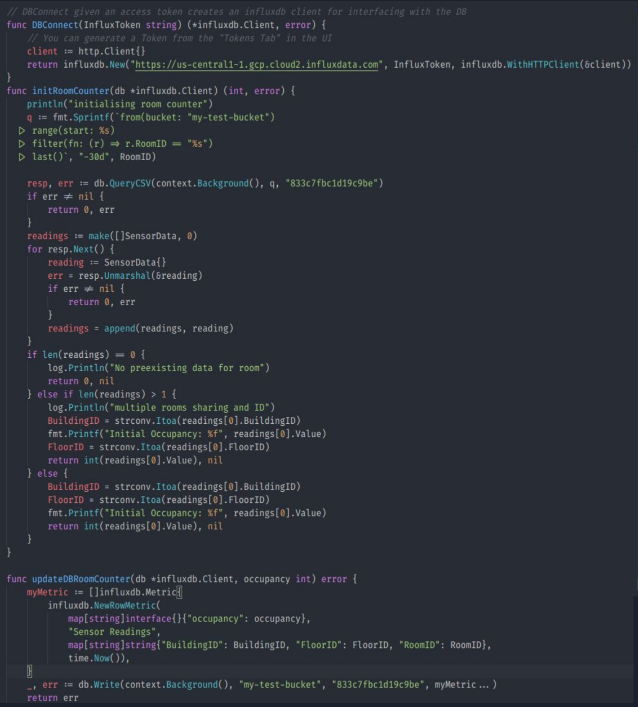


```
ubuntu@ubuntu:~/ROT$ sudo systemctl status pi-db.service
● pi-db.service - "Influx DB sensor logger"
   Loaded: loaded (/etc/systemd/system/pi-db.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2020-02-28 15:25:58 UTC; 4s ago
 Main PID: 1928 (pi)
    Tasks: 9 (limit: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

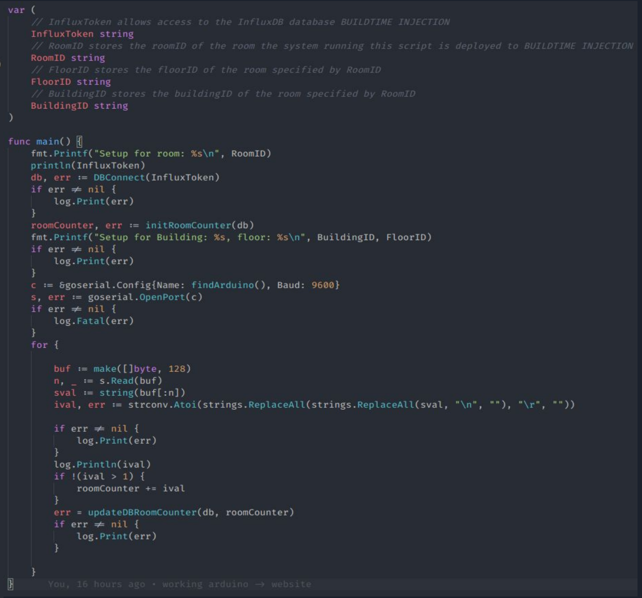
HiFi Prototype Component: Sensor Reading Relay: Code



```go
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.Print("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.Print(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"`
}
```



```go
// 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-central1-1.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")
  ▷ range(start: %s)
  ▷ filter(fn: (r) ⇒ r.RoomID == "%s")
  ▷ last()`, "-30d", 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.Printf("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.Printf("Initial Occupancy: %f", readings[0].Value)
        return int(readings[0].Value), nil
    }
}

func updateDBRoomCounter(db *influxdb.Client, occupancy int) error {
    myMetric := []influxdb.Metric{
        influxdb.NewRowMetric(
            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
```



```go
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.Print(err)
    }
    roomCounter, err := initRoomCounter(db)
    fmt.Printf("Setup for Building: %s, floor: %s\n", BuildingID, FloorID)
    if err ≠ nil {
        log.Print(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.Print(err)
        }
        log.Println(ival)
        if !(ival > 1) {
            roomCounter += ival
        }
        err = updateDBRoomCounter(db, roomCounter)
        if err ≠ nil {
            log.Print(err)
        }

    }
}      You, 16 hours ago • working arduino → website
```

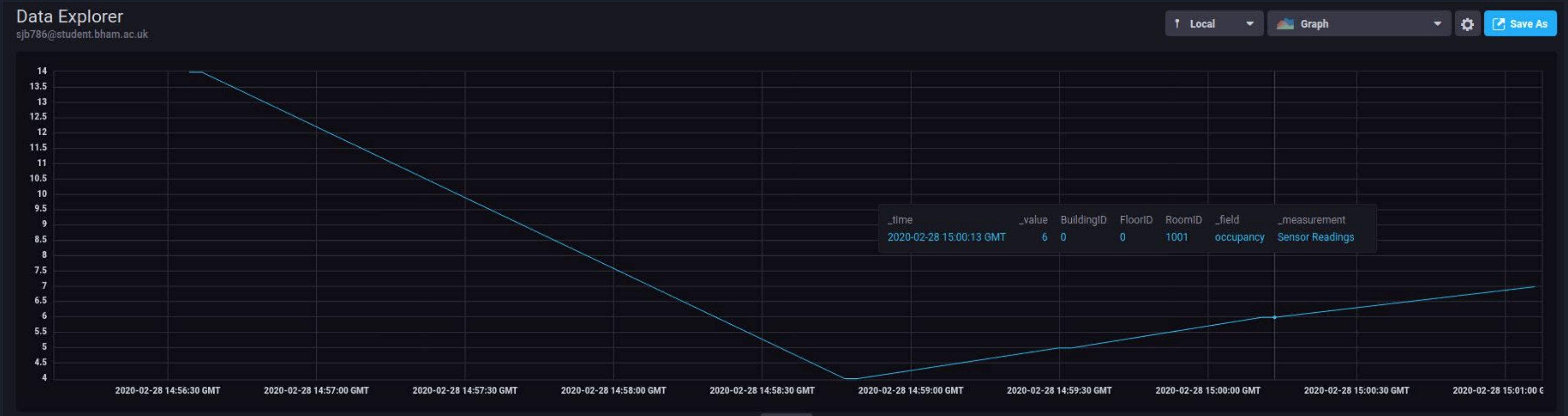
HiFi Prototype Component: InfluxDB

The backbone of any data-driven application is it’s 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.

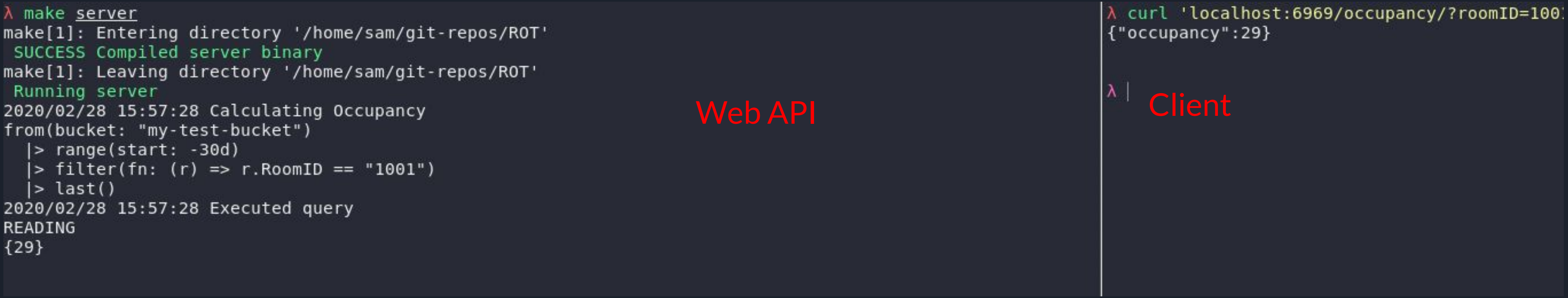


Data Explorer
sjb786@student.bham.ac.uk

_time	_value	BuildingID	FloorID	RoomID	_field	_measurement
2020-02-28 15:00:13 GMT	6	0	0	1001	occupancy	Sensor Readings

HiFi Prototype Component: Web API

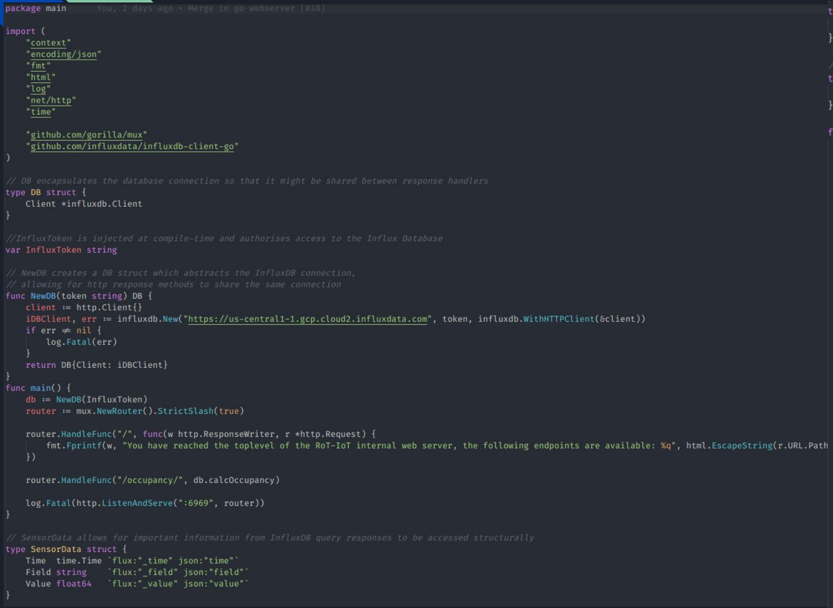
- As an intermediary between the database and our frontend web-app we have a simple web API written in Golang. This has an endpoint `*/occupancy/?{roomID|buildingID|floorID}*`
- Using this, upon loading a page that requires room occupancy information, a request is made based on the whether the occupancy 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.
-



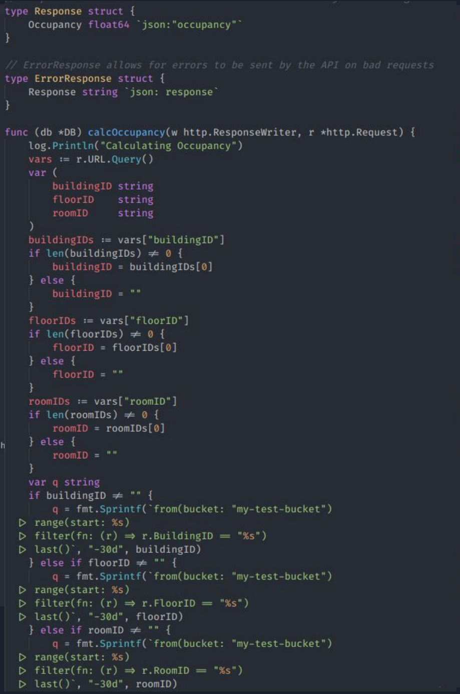
```
λ make server
make[1]: Entering directory '/home/sam/git-repos/ROT'
 SUCCESS Compiled server binary
make[1]: Leaving directory '/home/sam/git-repos/ROT'
 Running server
2020/02/28 15:57:28 Calculating Occupancy
from(bucket: "my-test-bucket")
  |> range(start: -30d)
  |> filter(fn: (r) => r.RoomID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

```
λ curl 'localhost:6969/occupancy/?roomID=100
{"occupancy":29}

λ |
```

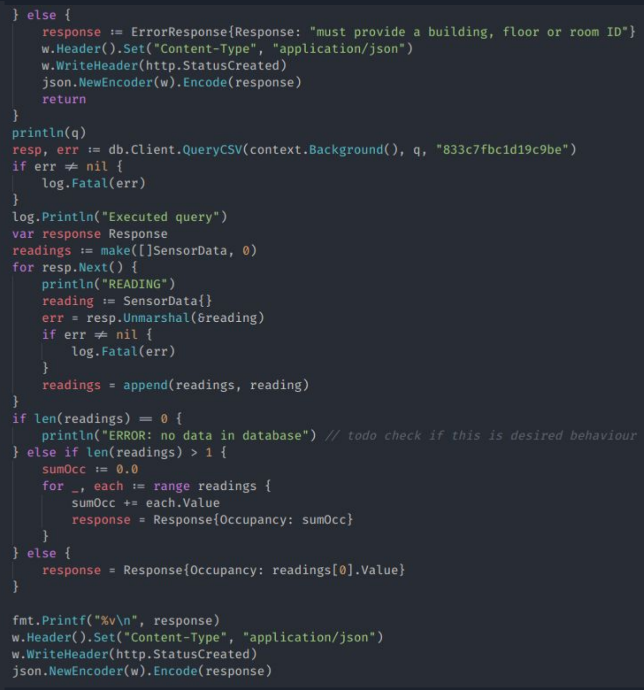
HiFi Prototype Component: Web API: Code



```go
package main      You, 2 days ago • Merge in go webserver (#38)

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 authorises 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-central1-1.gcp.cloud2.influxdata.com", token, influxdb.WithHTTPClient(&client))
    if err ≠ nil {
        log.Fatal(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 RoT-IoT internal web server, the following endpoints are available: %q", html.EscapeString(r.URL.Path
    })

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

    log.Fatal(http.ListenAndServe(":6969", 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"`
}
```



```go
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")
  ▷ range(start: %s)
  ▷ filter(fn: (r) ⇒ r.BuildingID == "%s")
  ▷ last()`, "-30d", buildingID)
    } else if floorID ≠ "" {
        q = fmt.Sprintf(`from(bucket: "my-test-bucket")
  ▷ range(start: %s)
  ▷ filter(fn: (r) ⇒ r.FloorID == "%s")
  ▷ last()`, "-30d", floorID)
    } else if roomID ≠ "" {
        q = fmt.Sprintf(`from(bucket: "my-test-bucket")
  ▷ range(start: %s)
  ▷ filter(fn: (r) ⇒ r.RoomID == "%s")
  ▷ last()`, "-30d", roomID)
```

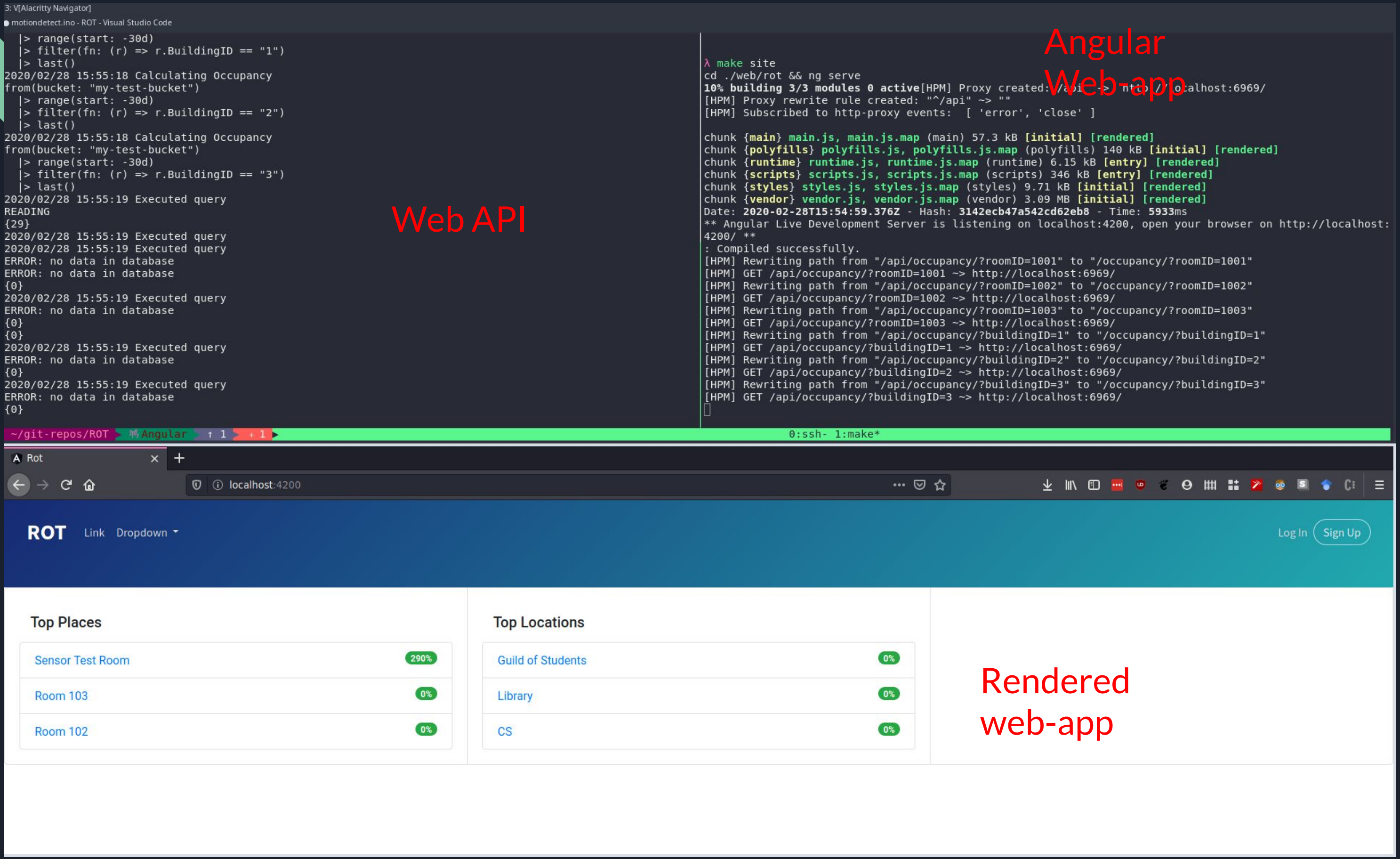


```go
} else {
    response := ErrorResponse{Response: "must provide a building, floor or room ID"}
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusCreated)
    json.NewEncoder(w).Encode(response)
    return
}
println(q)
resp, err := db.Client.QueryCSV(context.Background(), q, "833c7fbc1d19c9be")
if err ≠ nil {
    log.Fatal(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.Fatal(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.StatusCreated)
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.
- Each occupancy readout requires a request to the webserver for that specific roomID, floorID or buildingID. The resulting occupancy is then converted to a percentage using the *max_occupancy* field in the *locations.json* file.
- An improvement to this system may be to also host locations.json on the webserver allowing for easier update of locations in the future
-

HiFi Prototype Component: Frontend Web-App



```
  |> 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
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}
2020/02/28 15:55:19 Executed query
ERROR: no data in database
{0}
```

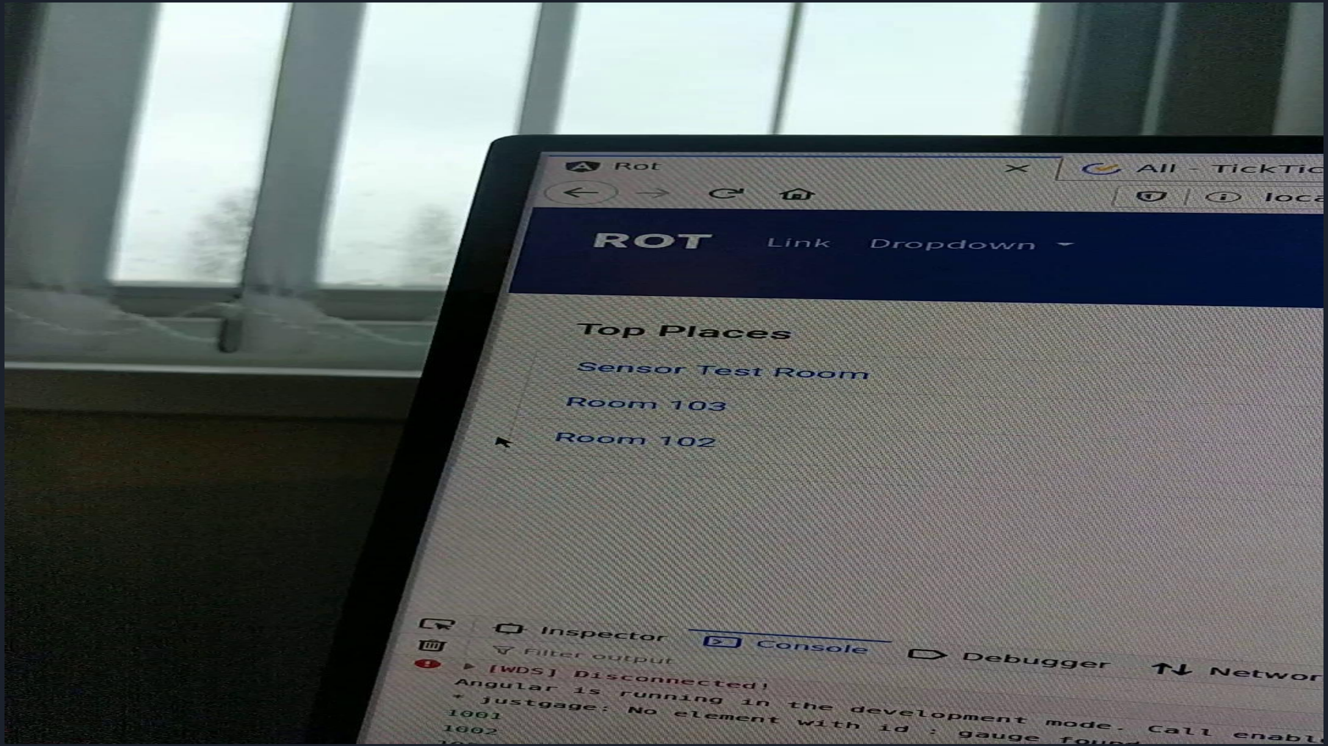
```
λ make site
cd ./web/rot && ng serve
10% building 3/3 modules 0 active[HPM] Proxy created: /api  -> http://localhost:6969/
[HPM] Proxy rewrite rule created: "^/api" ~> ""
[HPM] Subscribed to http-proxy events:  [ 'error', 'close' ]

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/
```

Top Places	
Sensor Test Room	290%
Room 103	0%
Room 102	0%

Top Locations	
Guild of Students	0%
Library	0%
CS	0%

HiFi Prototype: Full Stack Demo

Here you can see that the triggering of the sensor changes the occupancy reading for the ‘Sensor Test Room’ on the site, displaying the full prototype stack working together from sensor to web-app



```
▸ [WDS] Disconnected!
Angular is running in the development mode. Call enabl
+ justgage: No element with id : gauge found
1001
1002
```