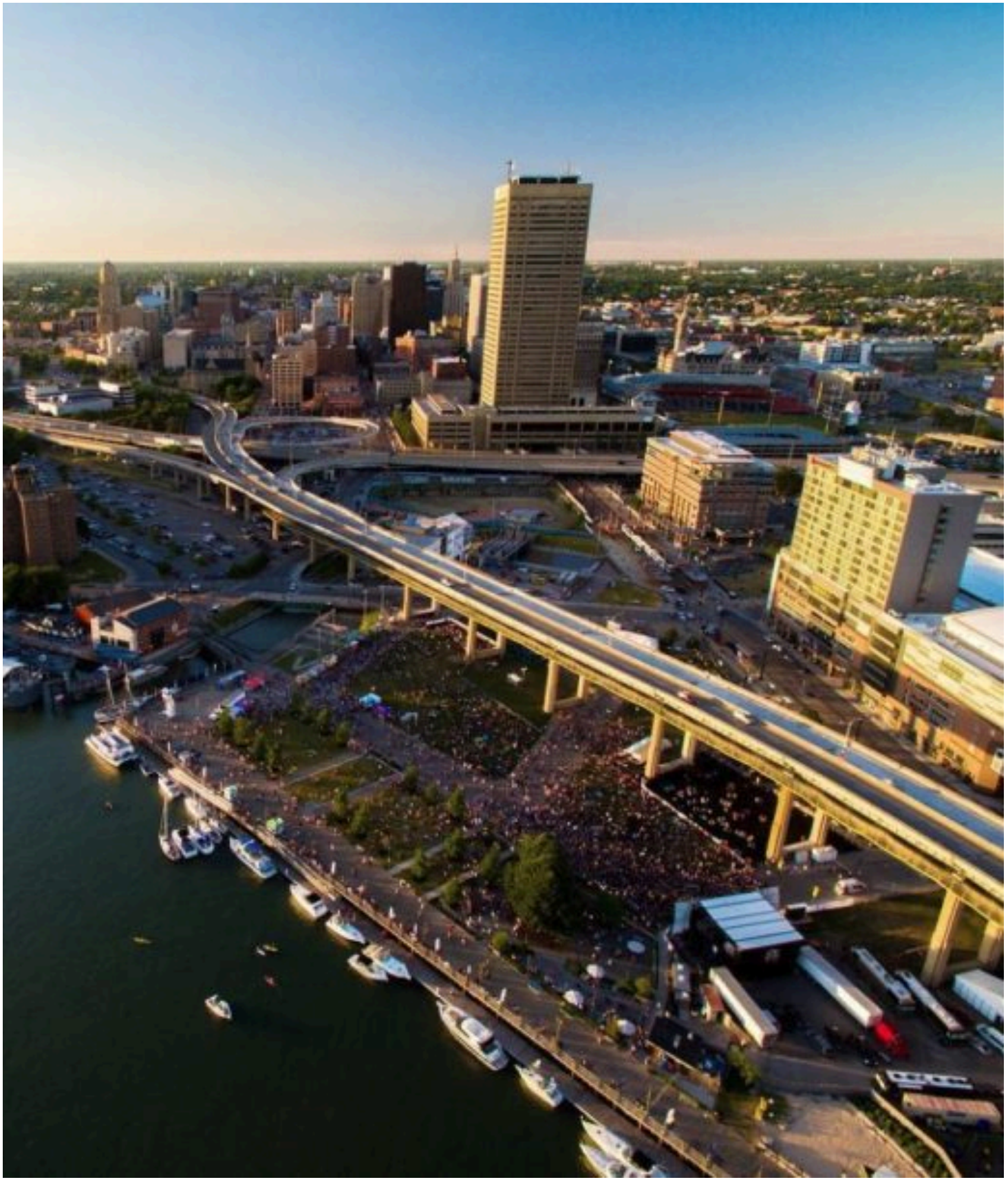
Smarter building, smarter health.



Introduction:

In this hackathon, our main focus is healthcare track, and the SMART Tech Hub.
Using the SMART Tech, we can use the available data to help enforce positive physical and mental health practices for employees as we start to reintegrate back into office spaces.
With this in mind, a fully functional smart city could improve employees' livelihoods via removing the stresses of commutes, meeting organisations, and even socialising in the covid world.
Our main assumptions are:

- There will be some sensors for tracking people (either lights, or wifi 6 hubs that allow us to see how many in a section of a room)
- CO2 or O2 monitors
- Card scanners which allows time stamping
- API for train rides
- Wifi extends outdoors
- Location of water fountain
- Scrum meetings are staggered allowing for organisation of work areas

By the end of the hackathon, we want to have an app prototype written in Java (as most of our team can use it), which would take mock input data (shown in the next table) from different sensors/data points from the building. To then be processed, and send notifications for the user to better help their mental and physical health.
The prototype should provide a smart alert if a user is working too long at their desk or coming up to lunch time. This notification would provide the user some options by giving them a list of other places to work for them to feel slightly refreshed, water fountains to keep hydrated or cafes/restaurants to eat at within the complex based upon distance and the amount of people there.
Further, the bot could show unpopulated seminary places for the user to make a scrum meeting or for sprint review, if the user requires a break or that the user needs a one on one meeting with individuals the application would then show quiet areas in their working area.
If we have enough time, the prototype could then help organise commutes via train for the user and show spots where the user can relax

in peace as well as integrating a covid symptom checker which in the future could integrate to a reporting tool that helps reduce the spread.

Market segment:

This application would predominantly target the employees of the tech hub initially. This is due to them already being moved, most likely to adapt to newer technologies and that the app is based around their work flow.
In the future, this could be extended to the other businesses, allowing the 23rd floor to be used to the fullest potential and allowing for easier socialisation.
With the secondary building opening, this current prototype will likely hold up since it relies heavily upon a modular design. These modules could pave the way for an AI system which handles the data more optimoly than what we had created.

If the city of buffalo develops the smart city tech correctly, this market could grow to the entire city. Meaning that individuals living within buffalo will never have to board a crowded train, deal with massive traffic jams and that businesses which adapt this tech, would likely have more customers as the app recommends them.

Test data

Data info	Raw Data	Use case
The key card swiping when entering a room should imprint the time of entry/exit. Completely theoretical.	[“ID-NUM” : ”TIME-OF-USE”], [“12 0000” : ”2020-10-13-13:01”] [“12 1331” : ”2020-10-13-13:04”] [“12 0000” : ”2020-10-13-18:01”] [“12 1331” : ”2020-10-13-20:01”]	By monitoring card usages, the app will know how many people per a floor, then can prioritise open spaces.
Monitor of CO2 or O2 levels on each floor in percentages. Completely theoretical. Either from smoke alarms to purpose built O2 monitors.	[“FLOOR-ROOM” : “O2-LEVEL”] [“AgileCeremonySpace12A” : “0.85”] [“FLOOR-ROOM” : “CO2-LEVEL”] [“KitchenettteA13” : “0.15”]	Having an increased O2 level/reduced CO2, could lead to improved problem solving and mental well being. Thus the app would alert the user if they're likely affected and advise them to move.
API, allows live feed of all trains/metros	[“START-END” : “ARRIVAL-TIME”] [“DISTRICA-DISTRICB: “14:05””] [“DISTRICA-DISTRICC: “14:15””] [“DISTRICA-DISTRICB: “14:25””] [“DISTRICA-DISTRICC: “14:35””]	Knowing when **all** employees are leaving from the onsite train station, the app can then organise their routes to reduce rush hour and thus reduce costs.
Lights, and or people sensor for different areas of the room	[“FLOOR-ROOM : POPULATION”] [“AgileCeremonySpace12A : 4”] [“AgileWorkSpace12A : 1”] [“KitchenettteA13: 12”] [“KitchenettteB10 : 3”]	Knowing where everyone is can allow the app to generate meeting locations on demand, and can allow prioritization of work spaces based on population

Outside of these sensors, we would also be using the current time from the users computer.

Below, will be the coding aspects of this app. With all sensors being their own object (allowing for newer sensors to be added), one main processing file (which reduces recalls to sub functions) and allows a user focused implantation.

The code isn’t proper, as development hasn’t started yet, this is for the developers to get an idea of what the bot will look like.

Coding ideas

CardData():

Variables:

Private Int _IDNumber

Private DateTime _lastCheck,

Functions:

```
getID():
    Return _IDNumber
setID(int ID):
    this._IDNumber=ID
getLastCheck():
    Return _LastCheck
setLastCheck(DateTime lastCheck):
    this._LastCheck=lastCheck
```

Constructors:

```
cardData(int ID, DateTime lastUse):
    setID(ID)
    setLastCheck(lastUse)
cardData(int ID):
    setID(ID)
cardData( DateTime lastUse):
    setLastCheck(lastUse)
cardData():
    Pass
```

airQuality():

Variables:

Private Double _airPercentage

Private String _Location

Functions:

```
getLocation():
    Return _Location
setLocation(string Location):
    this._Location=Location
getAirPercentage():
    Return _airPercentage
setAirPercentage(double amount):
    this._airPercentage=amount
```

Constructors:

```
airQuality(string Location, double percentage):
    setLocation(Location)
    setAirPercentage(percentage)
airQuality(double percentage):
```

```
        setAirPercentage(percentage)
airQuality( string Location):
        setLocation(Location)
airQuality():
        Pass
```

trainData():

Variables

Private string _route

Private datetime _arrivalTime

Functions

getRoute():

        Return _route

setRout(string route):

        This._route = route

getArrivalTime():

        Return _arrivalTime

setArrivalTIme(dateTime arrivalTime):

        This._arrivalTime = arrivalTime

Light/people sensors -- populationLocation()

Variables

Private String: _Locations

Private int _population

Functions

```
getPopulation():
        Return _population
setPopulation(int population):
        this._population=population
getLocation():
        Return _Location
setLocation(string Location):
        this._Location=Location
```

Constructors

populationLocation(string locations, int population)

```
        setPopulation(population)
        setLocation(locations)
```

populationLocation(string locations)

```
        setLocation(locations)
```

populationLocation(int population)

```
        setPopulation(population)
```

populationLocation()

        Pass

Processor():

Variables

```
Var cardData[] _cardData
Var trainData[] _trainData[]
Var populationLocation[] _populationLocation[]
Var airQuality[] _airQuality[]
```

Functions:

```
GenerateData():
set tempData to readData("keycard.txt")
tempData.forEach((IDNum, dateTimeOfUse) -> {
     Var cardData() tempCardData= new cardData(setID(IDNum),
     setLastCheck(dateTimeOfUse)
     _cardData.append(tempCardData)
})
Set tempData to readData("O2.txt")
tempData.forEach((FloorNum, airQuality) -> {

     Var trainData() temptrainData =new trainData(
     setLocation(FloorNum),
     airPercentage(airQuality))
     _trainData.append(tempTrainData)
})
Set tempData to readData("light.txt")
tempData.forEach((Location, population) -> {
     Var populationLocation() tempPopulationLocation = new
     populationLocation(
          setLocation(Location),
          setPopulation(population))
     _populationLocation.append(tempPopulation)

})
```

readData(string url):

```
Set data to []
Set tempData to <String, string> tempData = new
HashMap<String, string>();
//A dictionary
Set fileOpener to new File(url)
fileReader= new scanner(fileOpener)
while(fileReader.hasNextLine()):
     tempData = fileReader.nextLine()
     data.append(temData)
fileReader.close()
Return data.values()
```

getMostPopulatedSelction(string location, int currentFloor):

```
Set data to readData("people")
Set tempData and output to <>//same one as above
switch(location):
     case("kitchen"):
      Set tempData to data.keyset().removeIf(key ->
     !key.startsWith("Kitchenet"))
     break
     case("ceremony"):
     Set tempData to data.keyset().removeIf(key ->
!key.startsWith("AgileCeremonySpace"))
     break
     case("work"):
     Set tempData to data.keyset().removeIf(key ->
!key.startsWith("AgileWorkSpace"))
     default(): set tempData to data
if(currentFloor is greater than or equal to 1) :
     tempData.forEach((workLocation, population) -> {
          Set TestFloorNum to
worklocation.replaceAll("[^\\d.]","")
          Cast TestFloorNum to int and set it to floorNum
          if(floorNumber is greater than or equal to
currentFloor +/-1):
               output.append(workLocation,population)
     })
Else:
     Set output to tempData
Return output
```

getMostPopulatedSelction(int currentFloor):

```
Set data to readData("people")
Set tempData and output to <>//same one as above
if(currentFloor is greater than or equal to 1) :
     tempData.forEach((workLocation, population) -> {
          Set TestFloorNum to
worklocation.replaceAll("[^\\d.]","")
          Cast TestFloorNum to int and set it to floorNum
          if(floorNumber is greater than or equal to
currentFloor +/-1):
               output.append(workLocation,population)
     })
Else:
Set output to tempData
Return output
```

checkWorkTIme(dateTime currentTime, int alertReminderTime):

```
listOfUsersID = []
Foreach timeOfEntry in _cardData:
     if(timeOfEntry.key().addHours(alertReminderTime)
<currentTime):
          listOfUsersID.append(timeOfEntry.key())
Return listOfUsersID
```

checkO2(int floorNum, int o2level)

```
Foreach dataPoint in _airQuality:
     Int o2LevelFloorNum =
dataPoint.key().replaceAll("[^\\d.]","”):
     if(o2LevelFloorNum ==floorNum):
          if(dataPoint.data() < o2level):
               Return true
     Return false
```

Constructors:

processor():

generateData()

Main():

Variables

```
Const int o2level = 90
Int currentFloor =0
Processors mainProcess
```

Functions:

AlertUsers(string message):

```
    //GUI alert
```

CovidChecker():'

```
GUIoutput("Any new cough, lost in taste or smell?")
if(userInput ==False):
    GUIoutput("cdc further symptoms")
GUIoutput("Recommend that if they show symptoms , or have been
in contact with someone with covid. Self isolate and order
test")
```

checkTimes():

```
If (mainProcess.checkWorkTimes(now) is true):
    AlertUser("Heads up youve been working for a while, maybe
youd like a quick break to refresh? :)")
If(mainProcess.checkO2(currentFloor,o2level) is true):
    AlertUser("Quick heads up, 02 is getting low on the
floor, maybe get up and get some fresh air!")
```

Constructor:

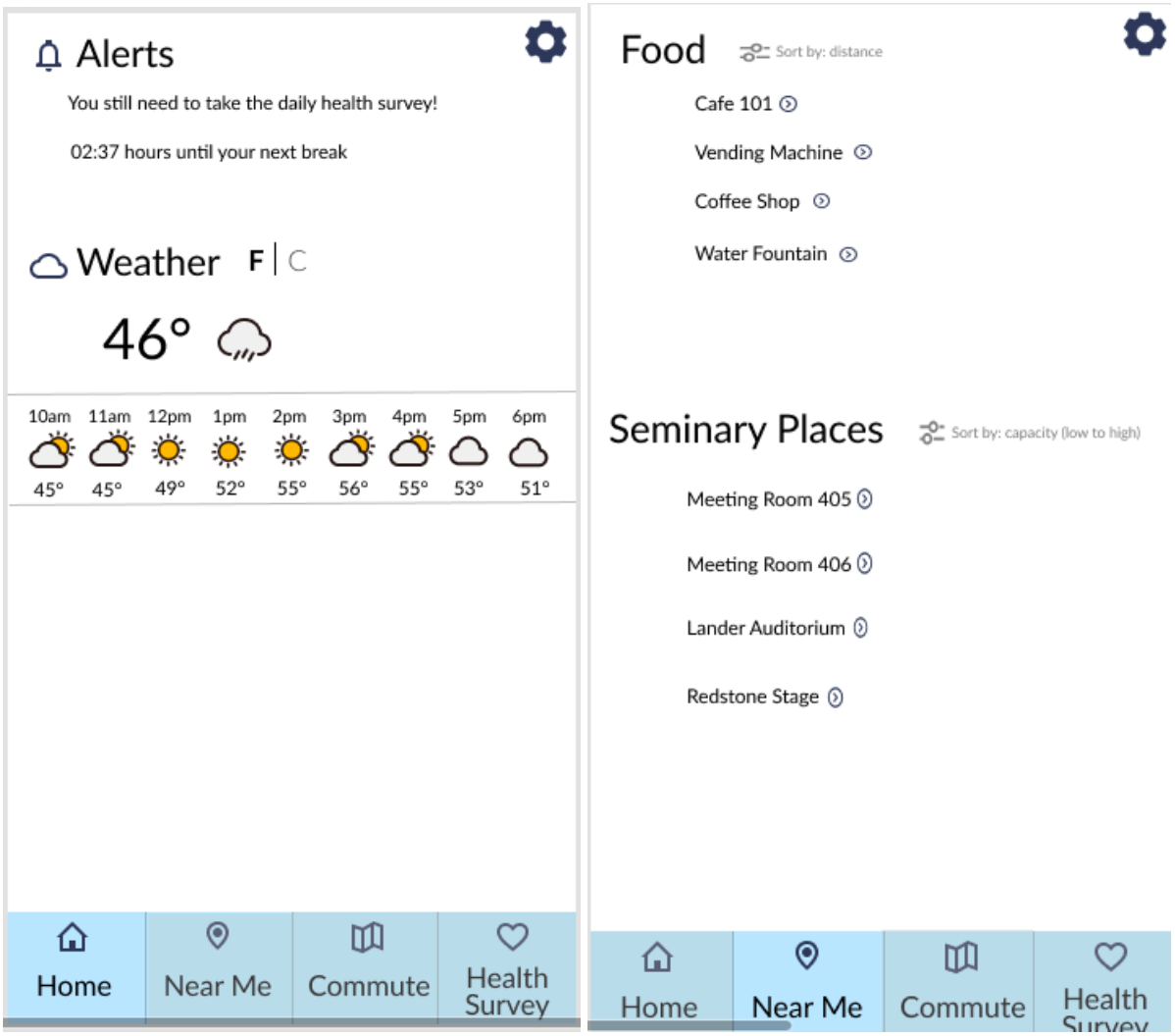
Main():

```
mainProcess = newProcess
While(True):
    //Some user input functionality, setting currentFloor
with the gui into the variable option
    switch(option):
    case("view most popular floors"):
        //Show
        mainprocess.getPopulationInfo.mostPopularSelection()
        .reverse()
    case("Select floor and room type"):
        //await user input of floor and room type
        //show
        mainprocess.getPopulationInfo.mostPopularSelection(u
        serInput)
    case("order train"):
    //a lot of complications that I'll not get done tonight
        case("covid wellness check"):
    Default: checkTimes();
```

Evaluation:

This weekend, we had developed this document and the simple prototype before you. With this you can see how a simple app could help manage common day to day tasks, along with some light improvements to help both physical and mental health.

The code within the prototype can allow rapid expansion, with sensor objects being added in the future and can be easily maintained with clearer documentation.
Below are some further mockups of what the app could become.



In the future, if we were to continue with this project. I believe within at least 5-10 sprints we could have a fully in depth app that would take full advantage of the building to improve the lives of the workers, and maybe even help make the building more efficient.

Within one year of development and continued investment both in the teams, and in the city, I could see this app being ported as part of a website, along with being ported to the phone for all people who would use the smart building.

However, we should learn from mistakes in the past.
Take Craigavon within Northern Ireland in the 1960s.
Ignoring the politics at that time, they had similar plans, building a new city that'll have a train station through their main work building, which would link into the shopping mall at the town center, that if you build it they will come mentality. This failed horribly.
Incorrect funding, and not enough businesses could stifle the innovation that the city would want or need and that simply building the latest tech will never be enough to attract.
We would need to take this project optimism, and ground it slowly in reality.

Overall, I look forward to your feedback!
Thank you for your time, Jamie.