sandwico

User Manual: Real-time Fire Escape Route

by ERP

Mathilda Bresler
u16313382@tuks.co.za

Pieter Braak
u16313382@tuks.co.za

Kateryna Reva
u17035989@tuks.co.za

Jason Louw
u16313382@tuks.co.za

Xiao Jian Li
u16099860@tuks.co.za

12 May 2019

University Of Pretoria, Hatfield
Engineering, Built environment and Information Technology

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Contents

1 System Overview

The Real-time Fire Escape Route (Real-time FER) System is a new approach to solve an age old problem. The interface is in the form of mobile application as well as desktop application. The main goal of the Real-time FER is to indicate to the agent using the application what the most efficient route would be to take in the case of an emergency. The system will perform this functions by anonymously tracking the building populations to be aware of the building's current state. It will be done with the use of a heatmap, which is generated using the Bluetooth sensors installed throughout the monitored location. This will show the quantity and distribution of the population throughout the building.

The escape routes are pre-planned by the building designers, since there are special considerations that need to be made, including which walls have fire proofing, and which areas should be avoided for safety reasons. The heatmap is then to be consumed by an AI algorithm to assign a fire escape route to each agent in the building. The building's population distribution and available routes must be taken into consideration when assigning a route to each individual.

With the building heatmap and time sensitive fire escape route in place, each agent in the building will be sent a push notification to his/her phone indicating what escape path to follow in case of emergency. A real time fire escape route system:

- A software system based on a windows server system.
- Android application and web application interface.
- Bluetooth sensor locating.
- System name: *RTFE System*
- System category:
  - *Major application*: performs clearly defined functions to achieve the system goal of providing evacuation from building at real time.
  - *General support system*: provides network support for a variety of users and applications on multiple platforms.
- Operational status:
  - Partially Operational
  - Under development
- Types of users:
  - Back-end users: Users responsible for maintenance, upkeep and general operation of the system.
  - Administrative users: Users responsible for managing the users of the system and the information access of various types of users.
  - Agents: Users that interact with the main interface of the application and are the main intended demographic of the application. These are the users that will be using the main functionality of the system.

2 System Configuration

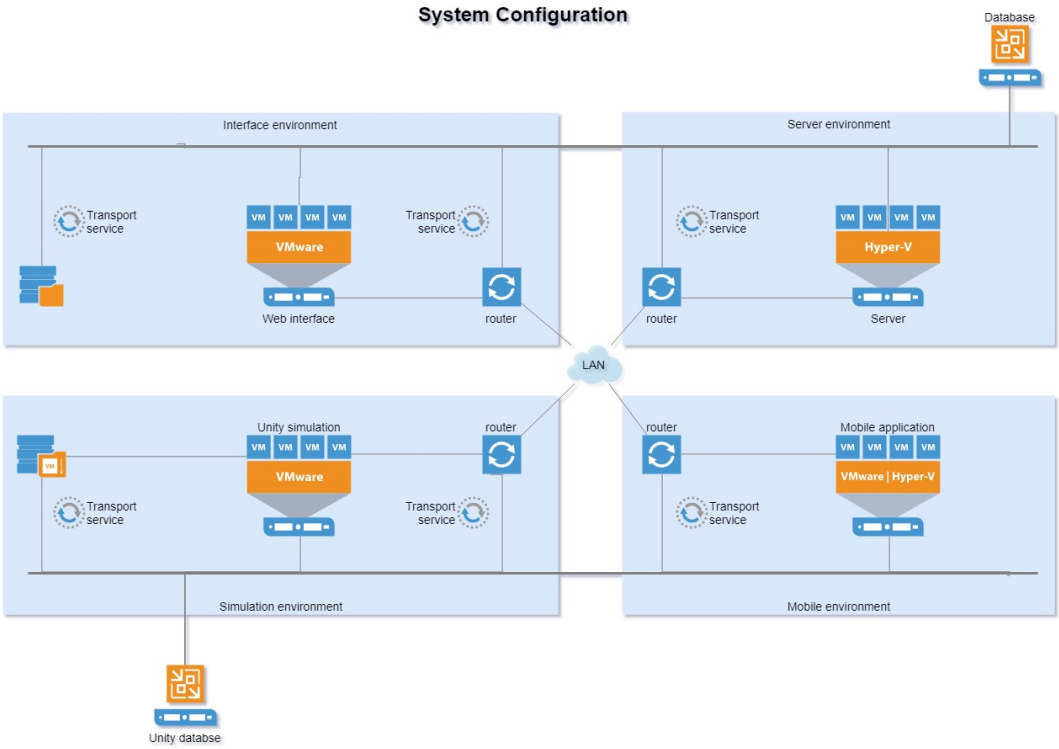


Diagram.jpg

2.1 Sensor configuration

(Unknown) Currently we do not have access to any sensors and can not explain how configurations of the sensors will take place.

2.2 Changing simulation parameters

(Planned) You can change which building will be used by the system in the simulation by changing settings in the browser. You will also be also be able to change how many people are in the building.

3 Installation

3.1 Installing the Application

Once a user is registered on the system they will be emailed details of their login details and a link to download the app. Alternatively for development and testing purposes the app can also be downloaded from our Git Page using this link: https://github.com/cos301-2019-se/Real-time-Fire-Escape-Routes.git

3.2 Installing the system locally

The system can be installed on a local machine by downloading and executing the jar file availible from this link https://github.com/cos301-2019-se/Real-time-Fire-Escape-Routes.git. If using this method you can run the system on any operating system.

3.3 Installing the system on the cloud

Currently the system doesn't allow this dynamic installation yet.

3.3.1 Setting Up Unity during development stage

- To run the simulation in unity you will need to download the unity engine from: https://unity3d.com/get-unity/download
- Please ensure you download the Unity 2019 version.
- Once that is complete you can you can create a git repository from our our GITHUB branch Unity: https://github.com/cos301-2019-se/Real-time-Fire-Escape-Routes.git
- Once you have unity installed and have cloned the github unity branch you will open unity and choose open project and choose the branch you just downloaded.
- You can then press the play button and it will trigger and api call to the server setup in 3.2 and the simulation will play out.

3.3.2 Setting Up Unity as a final product

- The final unity simulation will be exported as a functional executable that will not require prerequisites or game engines to run.A Simple double click will open the application and it will run the simulation, This will be part of future demos.

4 Getting Started

Before a user can access the system they must be registered by an admin in the browser the system will them email them their credentials along with a link to download the latest version of the application.

Take note if no admin is registered on the system yet a developer will need to add that user to the system so that he/she may add more users on their own.

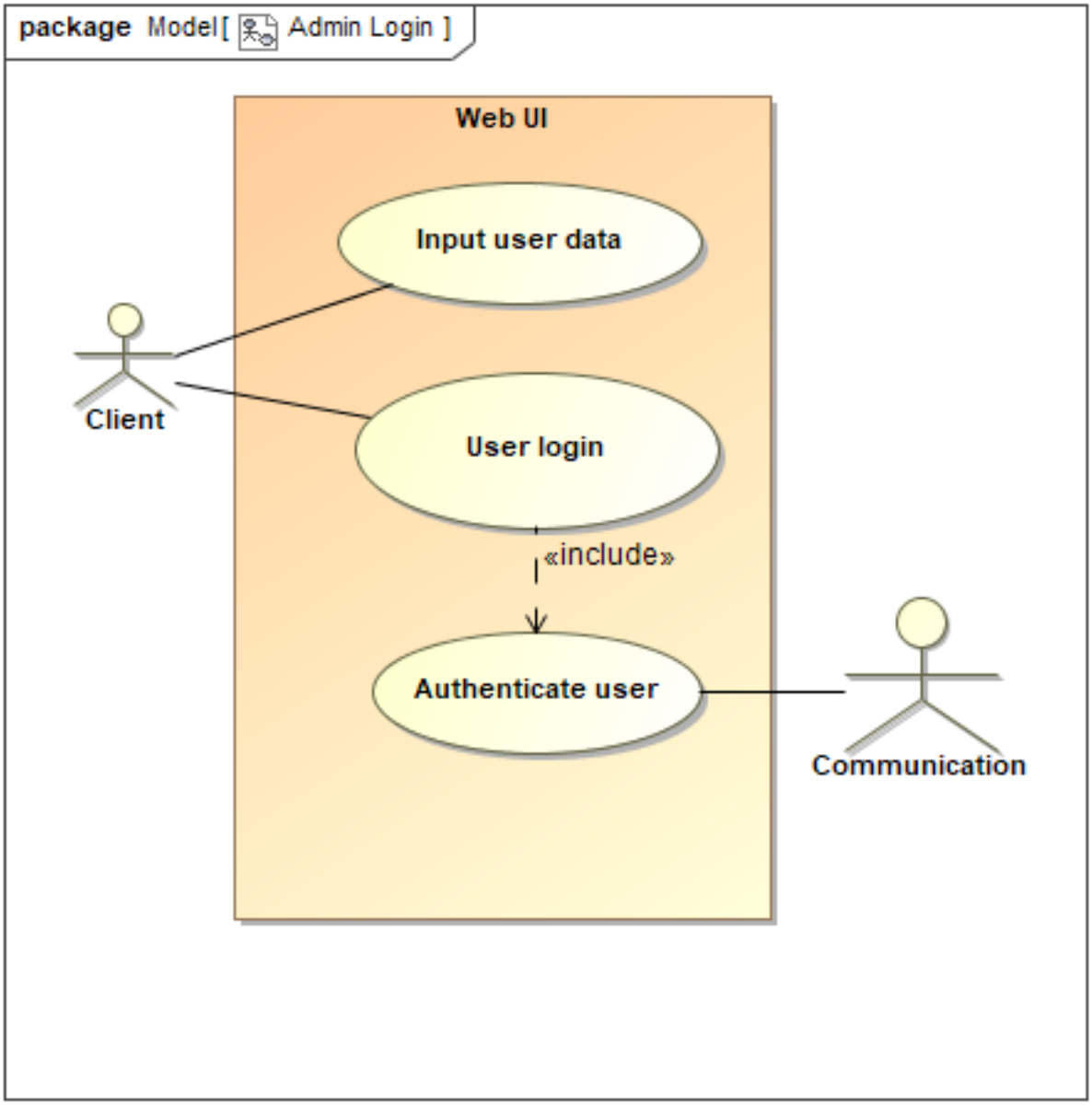
Once a user has been added to the system they can download the application and log in to the application. This gives them access to the main functionality of the system. When an alarm is triggered the system responds by performing the necessary calculations and then pushing the relevant information to the applications installed and holding active user accounts.
Once the user receives the information from the application they respond accordingly.

After the system has fully executed the evacuation procedure the data is backed up and the system is re-set, clearing the data on the application and reverting it to it's original state.

The application will exit once the user is removed from the database, this functionality is still to be implemented.

5 Using the system

- *Use case 1* - Administrative login to the system:



  - Accessing the browser: Assuming the system is installed locally you can access the system via http://localhost:8080/ or enter the IP of the PC where the software is installed on.
  - Inputting user details: The user inputs their personal login information into the appropriate fields.
  - Authenticating the user: The information is pushed from the web browser application to the system using the function *login(String name, String pass)* this then uses a post request, sending the data as a Json string in the format:

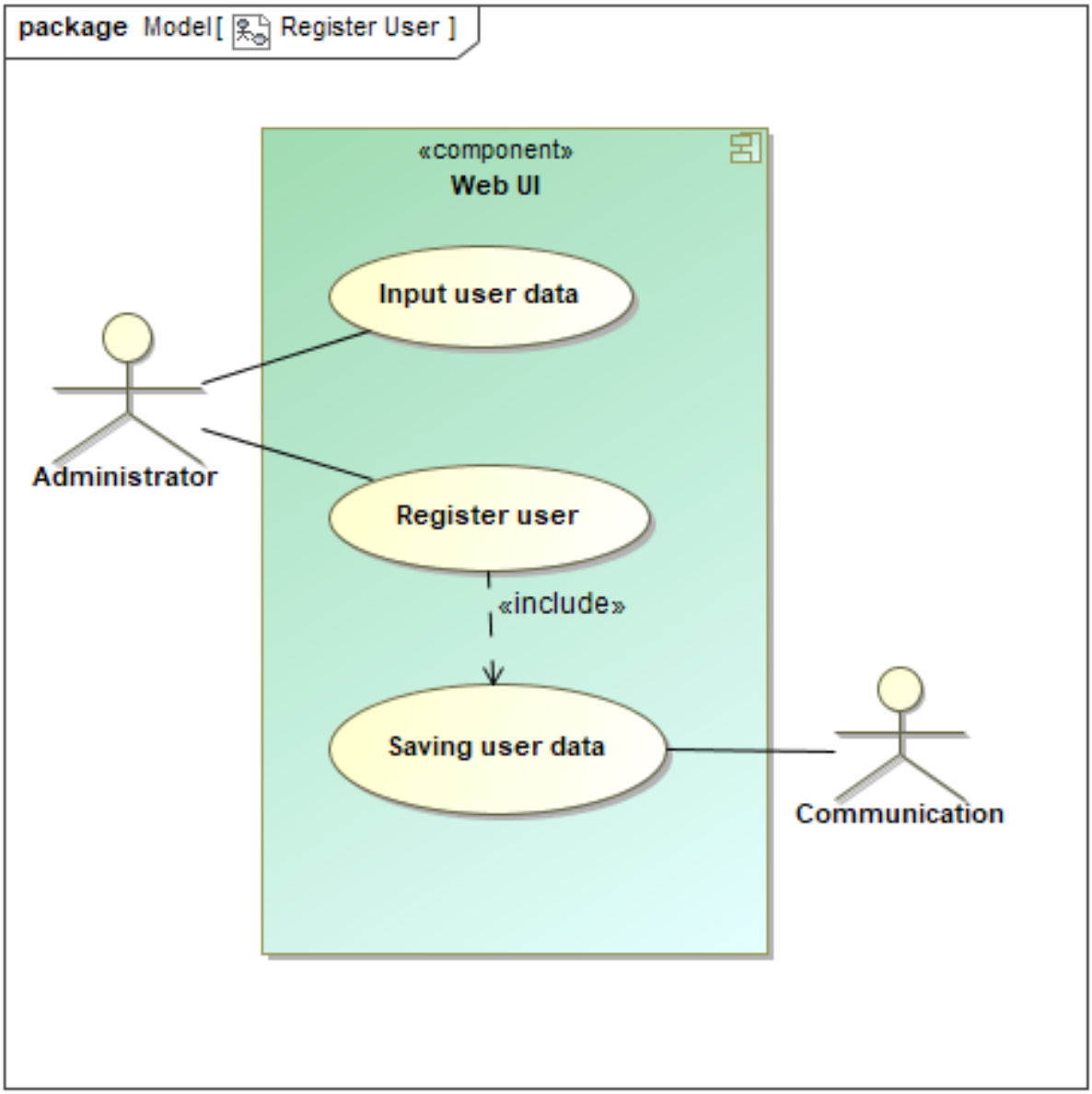
```
{                                                    1
    "type" : "login",                                2
    "name" : "John Doe",                             3
    "pass" : "12345"                                 4
}                                                    5
```

    , where this information is then compared to the information stored on the database using the function *search(name,pass)* contained on the HTTPServer. If the user exists in the database

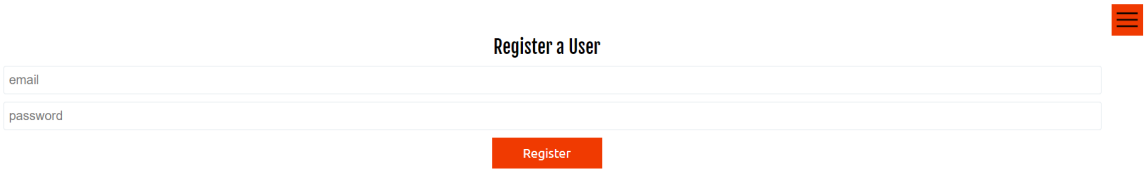
and their provided information is correct the user is logged in and a success message is returned from the server in the format

```
{
    "status": true,
    "msg": "Login success"
}
```

- *Use case 2* - Registering a new user on the system:



  - Accessing the browser: Assuming the system is installed locally you can access the system via http://localhost:8080/ or enter the IP of the PC where the software is installed on.
  - Administrative login: The user goes through *Use case 1* to gain necessary access to the system.
  - Inputting the new user information: The administrative user navigates to the page where they input the data for the new user (Agent) who is to have access to the mobile application system



  - Registering the new user on the database: The web application makes a post request to the HTTPServer containing the relevant user information as a Json string in the format:

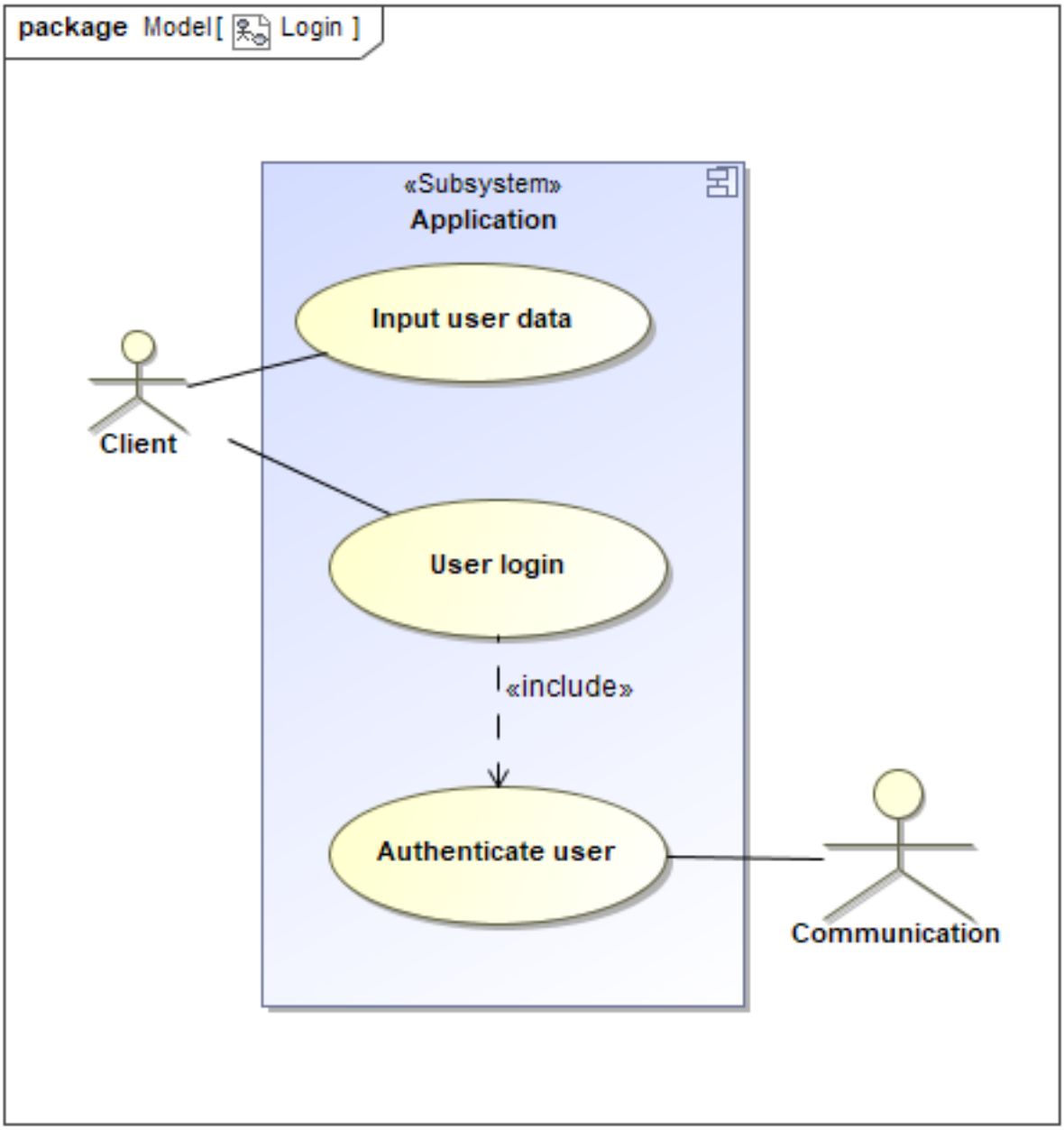
```
{                                                                    1
    "type" : "register",                                             2
```

```
    "name" : "John Doe",
    "pass" : "12345"
}
```

. The information is then processed through the function *write(name,pass)* and stored in the database. An appropriate success message is then returned to the web application in the format.

```
{
    "status": true,
    "msg": "User Successfully created"
}
```

- *Use case 3* - User (Agent) logging in to the mobile application:



  - Accessing the application: for that agent requires to have the Real-time FER application to be installed on the device.
  - Inputting user details: The user inputs their personal login information into the appropriate fields on the login page.
  - Authenticating the user: The information is pushed from the application to the system using a post request which sends the data in the format:

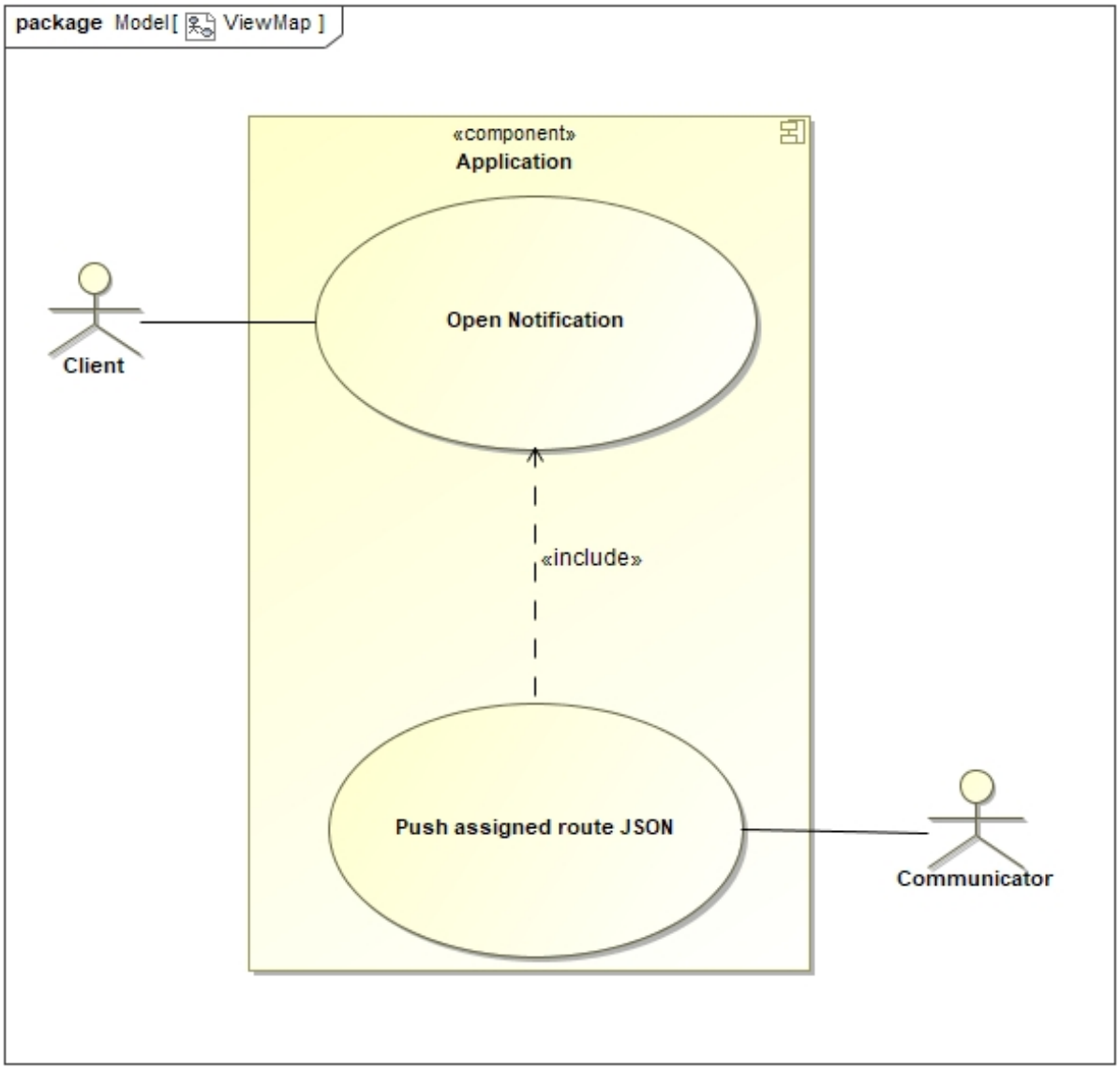
```
{
    "UserName": userName,
    "Password": password
}
```

    , where the information is then compared to the information stored on the database using the function *login(name,pass)* . If the user exists in the database and their provided information is correct they logged in, and an appropriate success message is then returned to the mobile application in the format.

```
{                                                    1
    "status": true,                                  2
    "msg": "Login success"                           3
}                                                    4
```

The application then responds to this information by displaying welcome page and will now be on stand-by-mode (running on the background).

– *Use case 4* - Viewing map on application:



- Receives notification

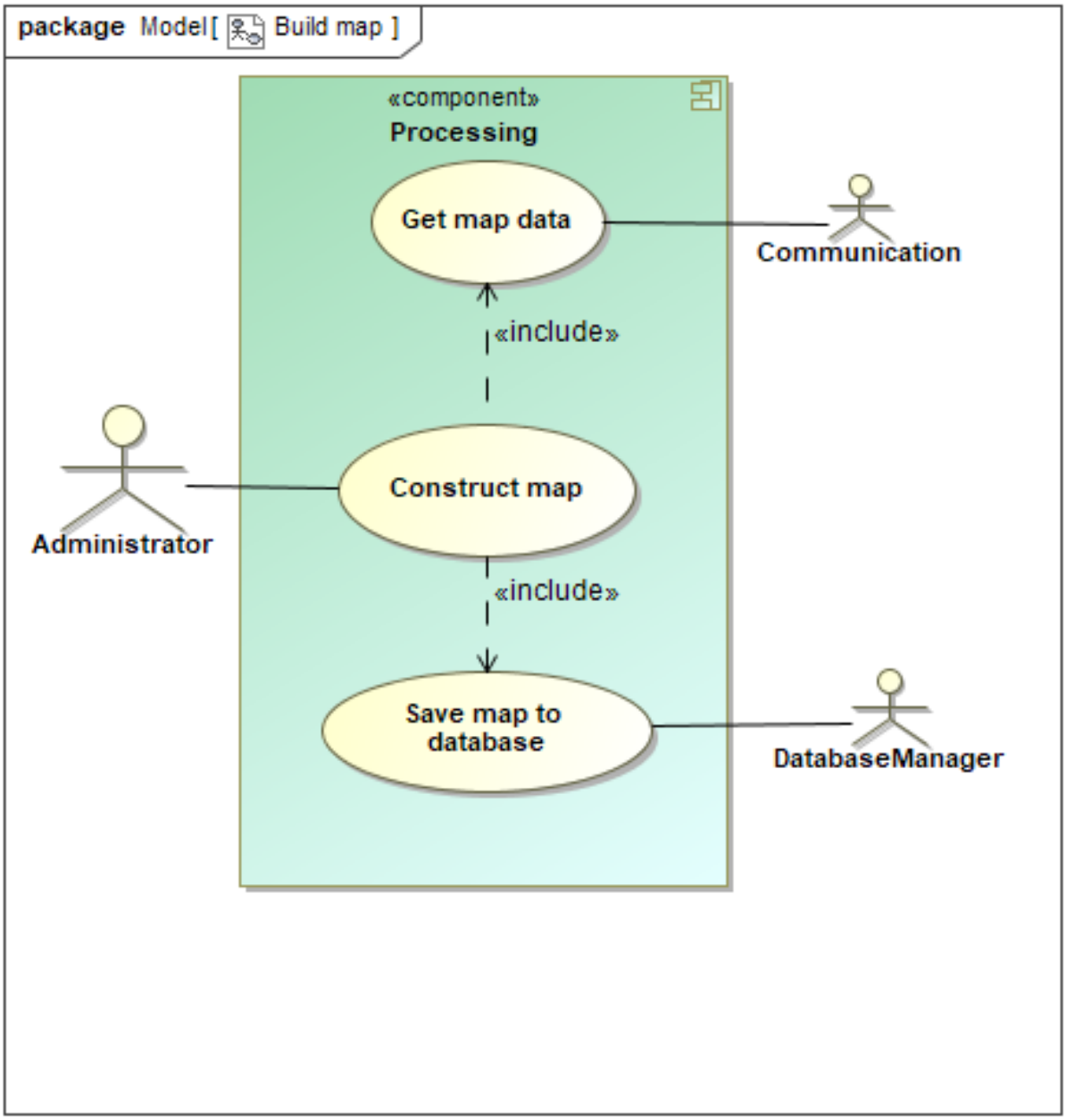
- Open Notification: Client opens notification that was send by the Communication.

- Push assigned route JSON: the JSON string with details about where in folders to find assigned route map was pushed onto application side in the following format:

```
{                                                    1
    "status": true,                                  2
    "folder": "Buildings",                           3
    "subfolder": "office1",                          4
    "image": "building.jpg"                          5
}                                                    6
```

- After image is retrieved from given path, that is in the form of JSON, the application will display the map on the user's device.

– *Use case 5* - Building a map:



∗ Sending the data to the server: A post request is made to the HTTPServer containing the data as a Json string in the format

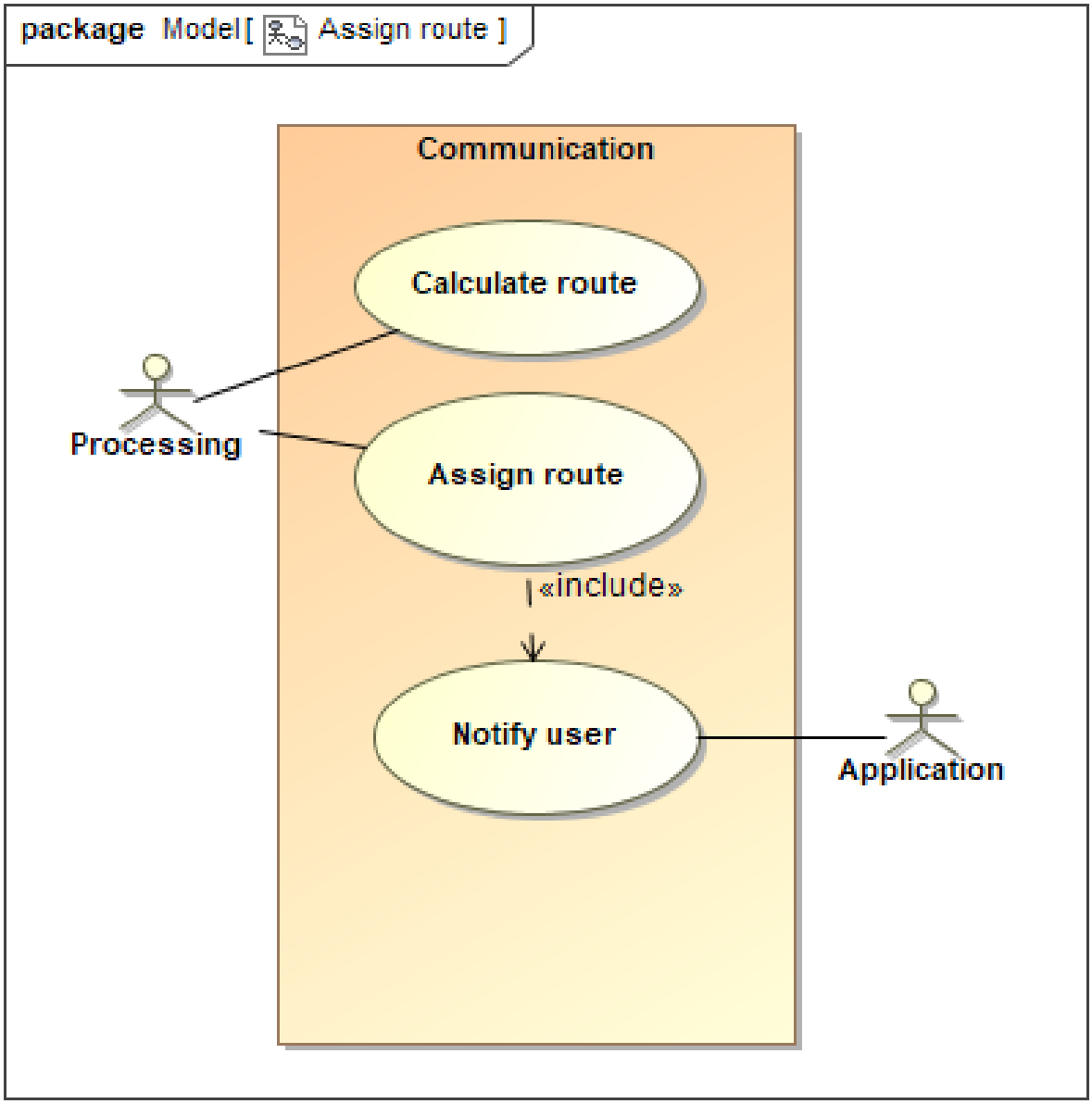
```
{                                                                   1
        "type":"unity",                                             2
        "floors":[                                                  3
                {                                                   4
                        "floor":0,                                  5
                        "corners":[                                 6
                                [0,0],[0,5.8],[4,10],               7
                                [24,10],[24,1.7],                   8
                                [17,1.7],[17,0]                     9
                        ]                                           10
                }                                                   11
        ],                                                          12
        "halls":[                                                   13
```

```
                {                                                       14
                        "floor":0,                                      15
                        "corners":[                                     16
                                [0,4.5],[0,5.8],[24,5.8],               17
                                [24,4.5],[13.3,4.5],                    18
                                [13.3,0],[12,0],[12,4.5]                19
                        ]                                               20
                }                                                       21
        ],                                                              22
        "rooms":[                                                       23
                {                                                       24
                        "floor":0,                                      25
                        "corners":[                                     26
                                [0,0],[0,4.5],[4,4.5],[4,0]             27
                        ]                                               28
                }                                                       29
        ],                                                              30
                                                                        31
        "doors":[                                                       32
                {                                                       33
                        "floor": 0,                                     34
                        "type": "buildingExit",                         35
                        "position": [12.6,0]                            36
                }                                                       37
        ],                                                              38
        "people":[                                                      39
                {                                                       40
                        "floor": 0,                                     41
                        "id": 0,                                        42
                        "position": [2,2]                               43
                },{                                                     44
                        "floor": 0,                                     45
                        "id": 1,                                        46
                        "position": [2,2]                               47
                }                                                       48
        ]                                                               49
}                                                                       50
```

* Processing the data: The server accepts the Json data and performs the following functions:
  - Data gets delagated to the BuildingGenerationAPI using the function "HandleReqeust(JSONObject request)"
  - Then the API makes use of the Builder Design Pattern to create the various parts of the building and add them into the final building object which contains the various functions for making escapes possible
  - The Builder Pattern makes use of a Director called "BuildingManager" that creates the appropriate concrete builders for each part of building's data that was sent throught the request.
* After the map has been constructed and saved the server can use this data to perform the necessary calculations.
* An appropriate success message is then returned to the calling function.

```
{                                                       1
    "status": true,                                     2
    "msg": "Building built successfully"                3
}                                                       4
```

- *Use case 6* - Pushing notifications to the application:



  - The server is alerted to an emergency: This consists of either a sensory trigger or a manual trigger. This uses the function assignPeople.
  - Processing of the data: The system performs a set of tasks to react to the emergency using the data from the sensors which are in the form of *X, Y coordinates*.
    * The sensors send the location data of each *Agent* to the system in the form of *X, Y coordinates*.
    * There is a function *addPerson* that uses the sensory data to find the room that each person is located in.
    * The rooms are then recursively traversed and each person in the room is assigned a route based on a heuristic calculation.
    * The system then checks to see if the building has been evacuated at certain time intervals.

- *Use case 7* - Running the simulation: This consists of a certain amount of steps
  - Starting up the simulation: This consists of opening up the Unity program and starting the Unity server.
  - Setting up the simulation environment: The HTTPServer sends the building data that was processed during *Use case 5*. The Unity server then uses this data to set up the simulation environment.
  - This simulation is then used to show the processes that the system performs.
- *Use case 8* - Removing a user from the system:
  - Accessing the browser: Assuming the system is installed locally you can access the system via http://localhost:8080/ or enter the IP of the PC where the software is installed on.
  - Administrative login: The user goes through *Use case 1* to gain necessary access to the system.
  - Inputting the new user information: The administrative user navigates to the page where they input the data for the user (Agent) who is to be removed from the database and the system, in the fields provided.
  - Removing user from database: The function *remove(name)* creates a post request to the HTTPServer with the data in the Json string format. The function then provides the necessary actions to delete the user from the database.
  - Disconnecting user from the system: Once a user is removed from the database they are disconnected from the system and their application exits.

6 Troubleshooting

Errors that might occur:

- Invalid building coordinates: Upon constructing the building coordinates are provided that indicate the corners of the room. These corners are considered invalid if:
  - They do not create a full room with connecting walls
  - Corner coordinates aren't exact

  The system handles these invalid point by:
- The system calls a check function first to see if the room is valid, if it isn't valid it will print a message in the console and it wont be added to the building
- If a room is valid but its coordinates falls outside the floorplan it wont add it to the building.
- Invalid user placement: A user's position is considered invalid if:
  - They are colliding with objects in the building such as walls.

  This will be handled once we have access to the sensors and can investigate the functionality of the sensors themselves.
- Network error: As of currently the servers are run locally. Once the servers are deployed online investigation will be done into the reliability of the system. Backup servers will be set up in case of primary server failure, with sufficient testing to ensure reliable service.