Resident Management Interface

Interface Identity

The Resident Management interface is where admin of our system will have the ability to manage current residents within our resident database. The admin will be able to add, fetch and delete current resident and admin data.

Resources

There are 6 different resources that are part of the Resident MAnagement interface:

- addResident(Resident: resident) Resident
- getResidentID(string: ID) string
- deleteResident(string: ID) string
- addAdminUser(Admin: admin) Admin
- getAdminID(string: ID) string
- deleteAminUser(string: ID) string

addResident

Syntax

```
addResident(name: String, roomNumber: int): Resident
```

Semantics

The addResident operation adds resident to the resident database.

Parameters:

- *Name* (String): Name of the Resident
- *roomNumber* (int): Room Number of the Resident

Preconditions:

- The *roomNumber* parameter must be an open room number.

Postconditions:

- Returns a Resident of the new resident.

Error Handling:

- 400 Bad Request: If the *roomNumber* parameter is not a valid room number, an error is returned.
- 500 Internal Server Error: Resident could not be added to the database

getResidentID

Syntax

```
getResidentID(name: String, roomnumber: int): String
```

Semantics

getResident retrieves residentID

Parameters:

- *Name* (String): Name of the Resident
- *roomNumber* (int): Room Number of the Resident

Preconditions:

- The Name and Room number must exist in the database

Postconditions:

- Returns the resident ID as a string

Error Handling:

- 400 Bad Request: If the name and room number do not match an existing resident.
- 500 Internal Server Error: Could not fetch ID from the database

deleteResident

Syntax

```
deleteResident(residentID: String): String
```

Semantics

deleteResident removes the resident from the database

Parameters:

- *residentID (String)*: Identifying string of resident

Preconditions:

- residentID must exist in the database

Postconditions:

- Returns the input ID string
- Resident no longer exist in database

Error Handling:

- 400 Bad Request: if residentID does not exist
- 500 Internal Server Error: Could not delete resident from the database

addAdmin

Syntax

```
addAdmin(name: String, password: String, role: String,): Admin
```

Semantics

The addEvent operation adds a new event to the calendar, taking a name, time, and date as descriptors for the new event.

Parameters:

- *name* (String): Name of new admin
- birthday (String): birthday of new admin
- *role* (String): role the admin is in (i.e. chef, doctor, etc)

Preconditions:

- The *name* parameter must be a string
- The *birthday* parameter must be a string

Postconditions:

- Creates adminID and returns admin

Error Handling:

- 400 Bad Request: The *name* parameter must be a string
- 500 Internal Server Error: Resident could not be added to the database

getAdminID

Syntax

```
getAdminID(name: String, password: String) : String
```

Semantics

The getAdminID will return the adminID with the matching name and role

Preconditions:

- Name and Role must be valid Strings
- Name and Role must match an adminID

Postconditions:

- Returns the adminID as a String

Error Handling:

- 400 Bad Request: If the name and role do not match an admin in the database.
- 500 Internal Server Error: Could not fetch ID from the database

deleteAdmin

Syntax

```
deleteAdmin(adminID: String): String
```

Semantics

deleteAdmin removes a specific admin from the database

Preconditions:

- adminID is a valid String

Postconditions:

- Returns adminID as a String

Error Handling:

- 400 Bad Request: If the adminID is not a valid ID.
- 500 Internal Server Error: Could not delete the admin

Data types and constants

Resident

```
interface Resident{
  name: string;
  RoomNumber: string;
  Pin: int;
  ID: String;
}
```

Admin

```
interface Admin{
  name: string;
  role: string;
  Password: String;
  ID: String;
}
```

Error handling

There are two main different errors that can be raised from multiple resources on the interface, a 400 Bad Request and a 500 Internal Server Error.

If an incorrect or unrecognized data type is passed through any of the resources, a 400 Bad Request error will be returned.

A 500 Internal Server Error occurs when there is some problem in the database when adding, deleting, or modifying an event.

Variability

There won't be too much variability since most users will have their accounts set up by the admin. Since the admins will be able to set up each user according to a guideling, there won't be any large changes in the interactions within our system.

Quality attribute characteristics

Performance

The resident and admin management interface will not use most of its components too frequently, however the get methods will likely be used the most out of all other interfaces. Due to this, the get methods will be designed to prioritize speed first while the add and delete methods will prioritize precision.

Reliability

It is essential that the resident management service is reliable because it is a service that allows all other services to be used. It must be accessible at all times and it must never provide false information or throw unhandled errors. Error Handling will be a priority since the integrity of this micro service is so important.

Rationale and design issues

The design of the Resident Management interface is guided by common practices and industry standards to make it easy for developers to understand and use. We will follow these practices when creating our Resident and Admin interfaces to make their use easily understandable and straightforward. This promotes consistency and avoids ambiguity within our interface to make sure the process does not hinder the user's experience.

We have also designed our system with modularity in mind. This is to improve the maintainability of our system and to allow for easy expansion down the line. With this design style kept in mind, we will be able to keep our interface reliable while trying to improve on the performance over time.

Usage guide

Here are some examples of how to use some of the resources in the resident management interface.

addResident

```
let newResident = addResident(“John Doe”, 33)
```

getResidentID

```
Const residentID = getResidentID(“John Doe”, 33)
```

deleteResident

```
const residentID = getResidentID(“John Doe”, 33)
Let removedID = deleteResident(residentID)
```

addAdmin

```
let newAdmin = addAdmin(“Mary Sue”, “password123”, “Director”)
```