This document details the various APIs that are needed to successfully setup a Pi on the customer's site. Before the setup begins, the customer should already have a **Tennant ID** and a **Tennant API Key** Otherwise calls to the APIs will throw errors at them.

1. Check Node Name

This API Verifies if a given node name is already in use or not.

Method: **`POST`**

Path: **`/api/v1/tennant/devices/setup/checkNodeName`**

Headers:

```
   x-tennant-id: string            // user's tennant id (required)
   x-tennant-apiKey: string        // user's api key (required)
```

Body:

```
  nodeName: string                 // the node name to be checked (required)
```

Response Example:

Status code = **200**

```
Body = {
         "isAvailable": false,    // will be true if the node name is available, false otherwise.
         "message": "The nodename 'pi2244' is already in use"
       }
```

Note: A status code of 403 will be returned with a failed error message if no tennant id or tennant api key were provided or if the provided values were invalid.

2. Add New Device

This API registers a new Pi Device. After checking the node name, you'll need to generate the ssh keys and call this api with that node name and the ssh key so that it is saved.

Method: **POST**

Path: **/api/v1/tennant/devices/setup/add**

Headers:

```
  x-tennant-id: string            // user's tennant id (required)
  x-tennant-apiKey: string        // user's api key (required)
```

Body:

```
 nodeName: string           // the node name to be checked (required)
 authKey: string            // generated public key that will be used by the container (required)
 bluetoothName: string      // the device bluetooth name (optional)
```

Response Example:

Status code = 200

Body =
```
{
    "message": "Device added",
    "deviceId": "1234567890"
}
```

Note #1: A status code of 403 will be returned with a failed error message if no tennant id or tennant api key were provided or if the provided values were invalid.

Note #2: A status code of 400 will be returned with a failed error message if the node name already exists. So only call this API after the check has passed.

3. Get Tunnel Parameters

This API Returns the parameters that are needed to set up the reverse ssh tunnel with a container.

Method: **`GET`**

Path: **`/api/v1/tennant/devices/tunnelParams`**

Headers:

```
    x-tennant-id: string              // user's tennant id (required)
    x-tennant-apiKey: string          // user's api key (required)
```

Response Example:

Status code = **200**

```
  Body = [
           {
             "tunnelHost": "<container-name>.azureservices.au.net",
             "tunnelPort": 59223,
             "tunnelUser": "mygym-dispatcher",
             "tunnelPassword": "xy7@9#B35ij?5",
           }
         ]
```

Note #1: The body is an array of containers because we might have two or more containers added in the future.

Note #2: A status code of 403 will be returned with a failed error message if no tennant id or tennant api key were provided or if the provided values were invalid.

4. Update Device

This API Updates an existing device that has been saved. For instance, we might want to update the node name or public key being used to access the Pi.

Method: **PUT**

Path: **/api/v1/tennant/devices/setup/update/{deviceId}** // deviceId or node-name

Headers:

```
 x-tennant-id: string           // user's tennant id (required)
 x-tennant-apiKey: string       // user's api key (required)
```

Body:

```
 nodeName: string          // the node name to be updated (optional)
 authKey: string           // generated public key to be updated (optional)
bluetoothName: string      // the device bluetooth name (optional)
```

Response Example:

Status code = **200**

```
Body = {
          "message": "Device updated"
       }
```

Note: A status code of 403 will be returned with a failed error message if no tennant id or tennant api key were provided or if the provided values were invalid.

5. Delete Device

This API Deletes an existing device that has been saved.

Method: `DELETE`

Path: `/api/v1/tennant/devices/setup/delete/{deviceId}` // deviceId or node-name

Headers:

```
    x-tennant-id: string              // user's tennant id (required)
    x-tennant-apiKey: string          // user's api key (required)
```

Response Example:

Status code = **200**

```
Body = {
           "message": "Device deleted"
       }
```

Note: A status code of 403 will be returned with a failed error message if no tennant id or tennant api key were provided or if the provided values were invalid.

Here are the proposed steps:

- Make a call to the **Check Node Name** API to verify if a given node name exists.
- If the API returns *isAvailable = false* (meaning node name is available), then generate the ssh keys for the Pi and authorize the key.
- Use the node name and the generated public key to call the **Add New Device** API so that the device is saved.
- If that succeeds, call the **Get Tunnel Parameters** API to obtain the container details that will be used to setup the tunnel.
- Setup a reverse ssh tunnel with the container using the parameters obtained from the **Get Tunnel Parameters** API call.
- Call the **Test Setup** API to verify that the setup was successful. After calling this API, the web app will talk to the container to trigger the Open.sh script which should open the lock if the setup was successful. If something went wrong, then we have two options to resolve the issue.
    - Call the **Update Device** API to update the device's parameters and try again, or
    - Delete the device completely using the **Delete Device** API and setting it up from scratch.