# GAMMA Project status report

**Contents**

[**GAMMA Project status report**](#_457wnyte3763) **1**

[**Front end**](#_ry9vqa1m825p) **2**

[Designing of GAMMA entities](#_hng81x1186ry) 2

[Web based code editing](#_grtr47psznwn) 7

[**Back end**](#_gg2kc650cxxj) **7**

[MongoDB access](#_tb6hysc1z3g5) 7

[API development](#_7ewto6kpk40s) 8

[Code generation](#_bug3artyo31k) 8

[DB import/export functions](#_tr8p6kqjcra) 8

[**Demonstration**](#_28khs38dxzza) **10**

[Gerade office](#_ogto655ptcp9) 10

[Iotiq office](#_pj6dzjktte6k) 10

[**Device services**](#_9093dmnm9ro3) **10**

[Door sensors](#_h1pzel6mvrfr) 10

[Temperature/Humidity/Pressure sensors](#_xj0f4eewf8pi) 10

[Blind controller](#_9cp56cib4qxu) 11

[Wall plug](#_aoadk4j5d3cg) 11

[Orchestrator service (binary coupler)](#_42bzbdd4p6ba) 11

[**Communication services**](#_ifadqgv078g) **12**

[Zwave](#_yjnq7por8psp) 12

[Zigbee](#_916nlrq23y4f) 12

[**Deployment**](#_wdqh4cowucyo) **13**

[Creating a bundle](#_utah1ue8qu6t) 13

[System deployment](#_mj05qvh1bhdj) 13

[**Operation**](#_8p1l6mbs3d7d) **15**

[Floor design](#_ims8syh76wlj) 15

# 

# Front end

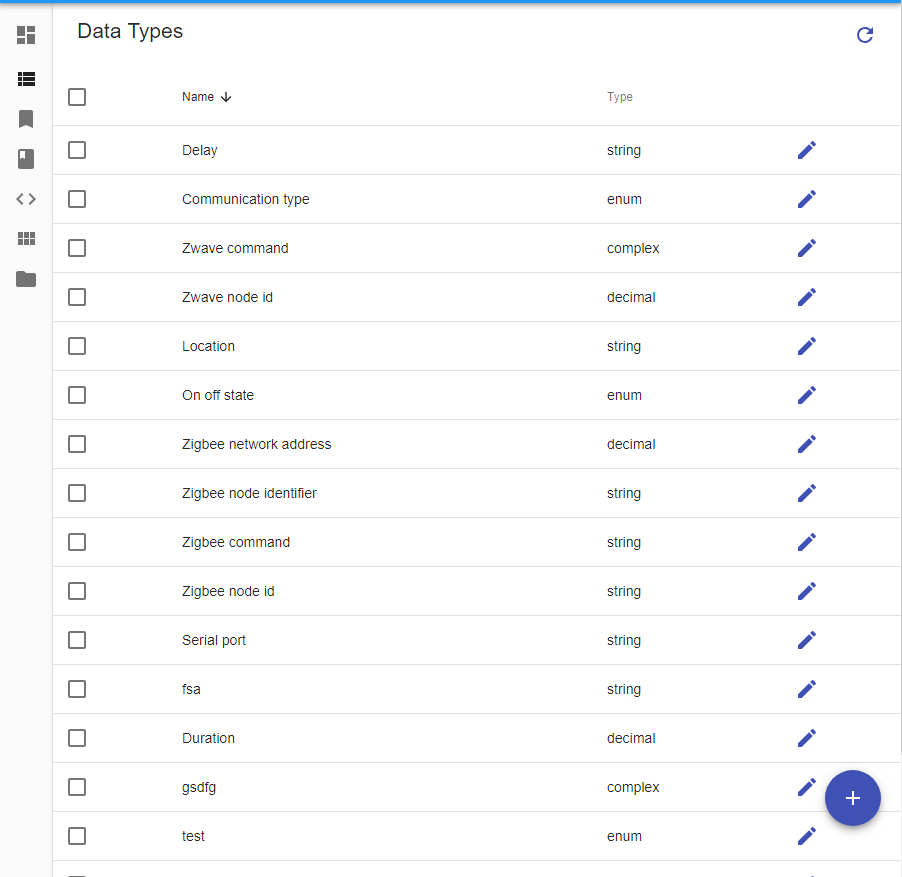
### Designing of GAMMA entities

We used react-admin framework for the front end side. This is a open sourced framework built on REST APIs, ES6, React and Material Design technologies. Its former name was “admin-on-rest”. It has features like custom actions, data validation, conditional formatting, List, Edit and Create pages and etc.

In our application, there are five major GAMMA entities; Data Type, Feature, DataPoint Type, Service Type and System. For every entity, there are three main Pages; **List, Create** and **Edit.** Also there is a Dashboard Page for a summary of our entities. It show how many Data Types and Feature we have and for the Service Type and System we can easily navigate from this main page.

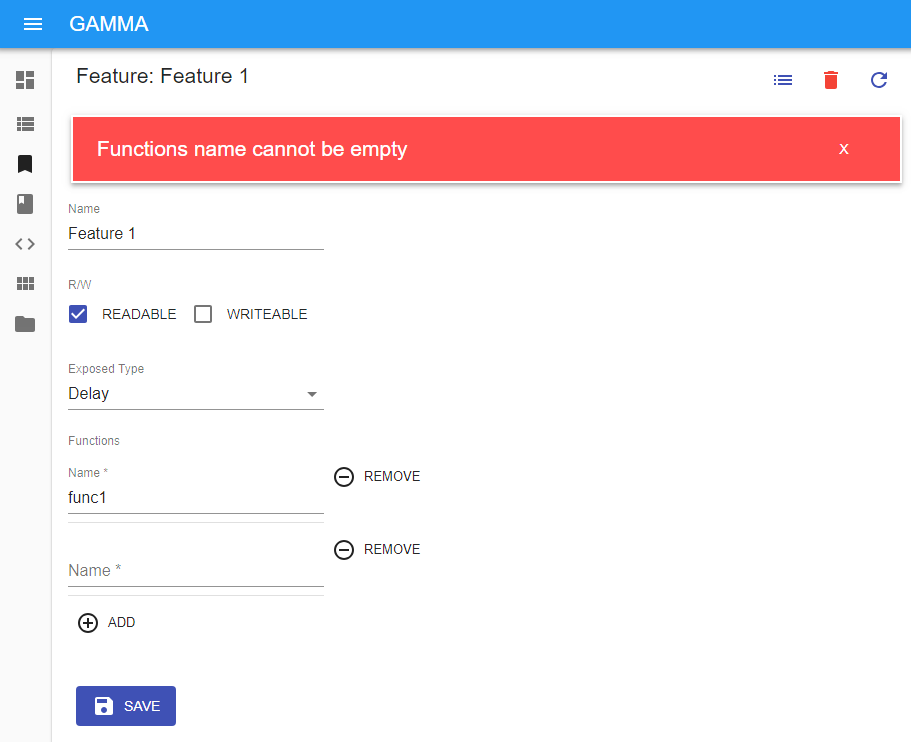
* + 1. DATA TYPE

Each Data Type must have two main properties: **Name** and **Type**. According the **Type** property, there will be different areas to enter. In GAMMA there are four Type of Data Type and these are *String*, *Enum*, *Decimal* and *Complex.* Every Type have its own constraints like Enum Data Type must have at least one value.



* + 1. FEATURE

A Feature have 4 properties: **Name**, **Readable/Writeable**, **Exposed Type** and **Functions. Exposed Type** refers to Data Type entity and a feature can have multiple functions. Also a feature can be only readable or writable at the same time. It cannot be both.

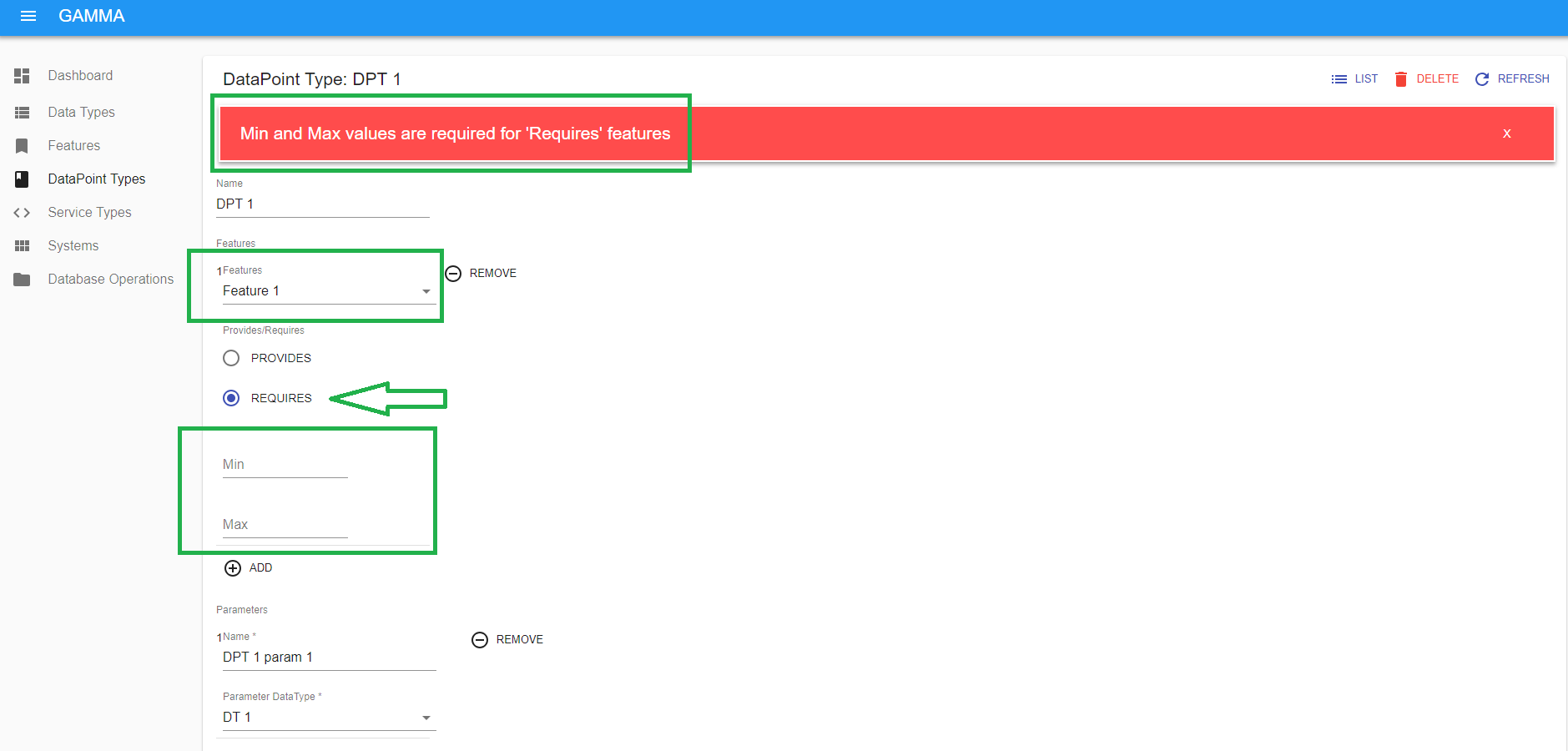


* + 1. DATAPOINT TYPE

A DataPoint Type have **Features**, **Parameters** and **Functions.** TheFeature of DataPoint Type must be *Provide* or *Require*. *Provide* means that this DataPoint Type can provide this feature to the other DataPoint Types and *Require* is opposite the *Provide* so it means that this DataPoint Type requires this feature. If a feature is required, *Min* and *Max* values is required too. These values means that how many device is needed for that feature.

DataPoint Types can have **Parameters** and every **Parameter** must have name and Data Type. Values of these parameters will assign in the Service Type or System stage.

Functions are like in the Feature stage.

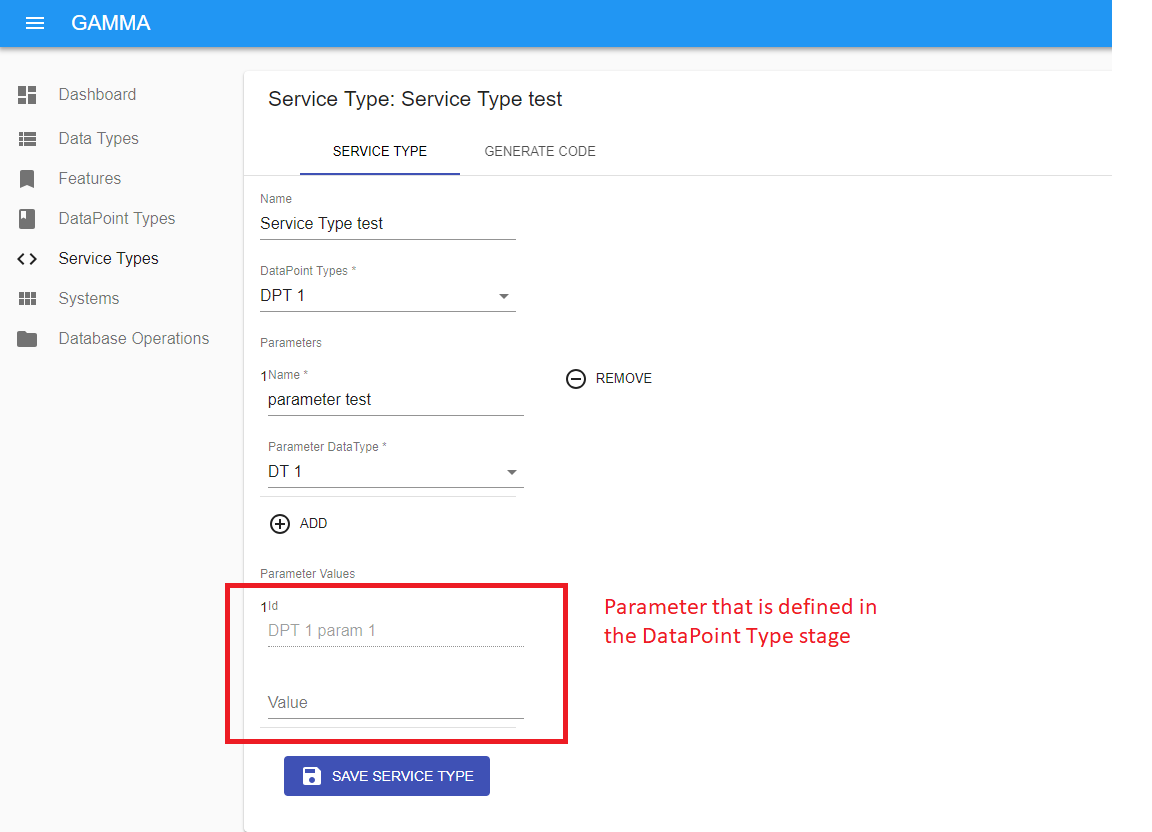


* + 1. SERVICE TYPE

Every Service Type have **DataPoint Type**, **Parameters** and **Parameter Values. DataPoint Type** refers to the DataPoint Type entity. **Parameters** are like in the DataPoint Type stage, each of them must have name and Data Type.

Parameter Values are the variables that was defined in the DataPoint Type stage but in the Service Type stage, user can define the values of them. It’s not obligatory to define in this stage because it can be defined in the System stage too.

Besides, in the Service Type stage, there is a Code Editor in a different tab (It’s only active in the Edit mode, not in the Create mode) Details of that will be in the “Web Based Code Editing” section



* + 1. SYSTEM

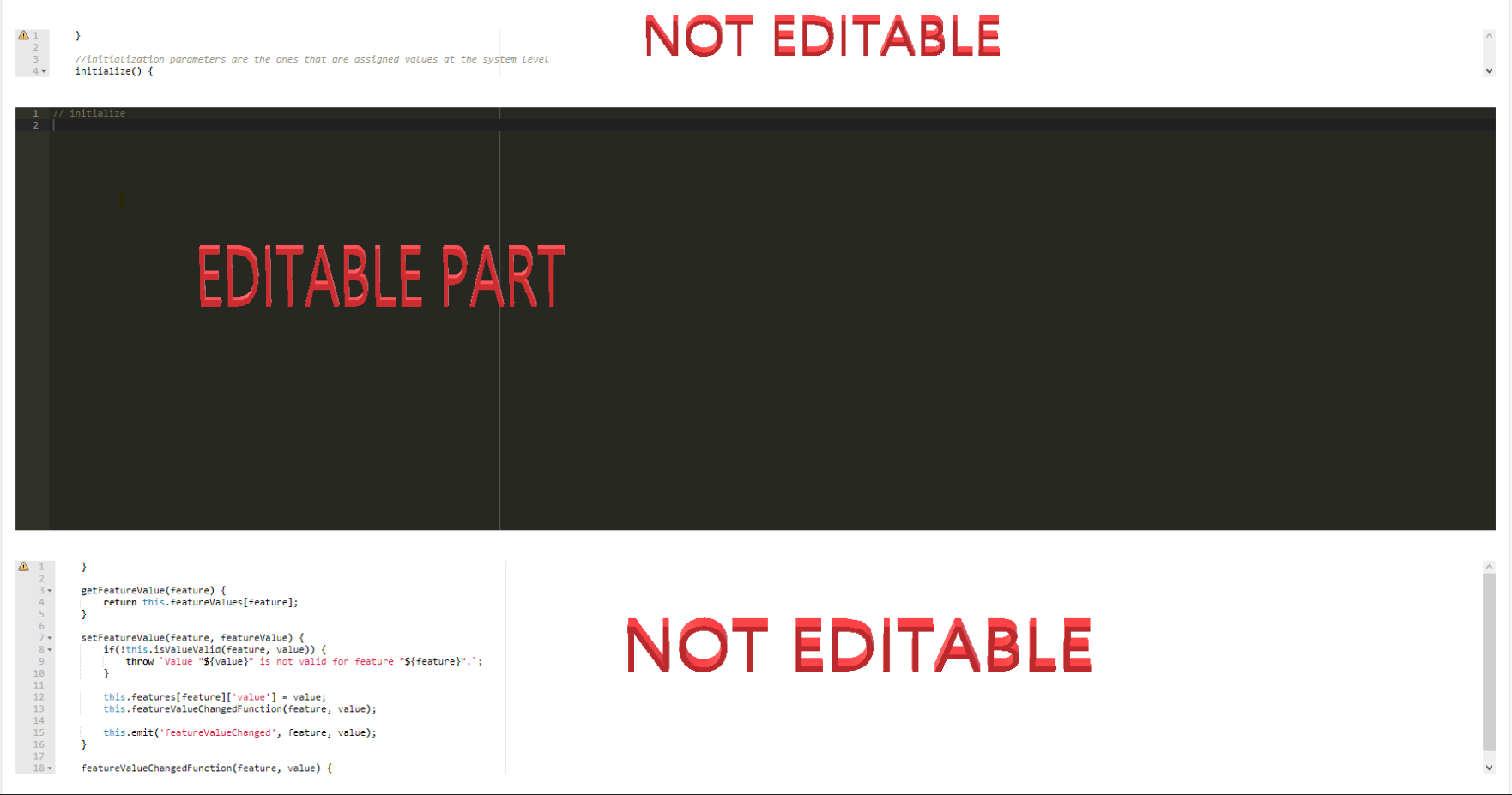
A System have **Name**, **Services**, **Service Relations** and **Parameter Values** properties. Services refers to Service Type entity and it consists name and service type.

**Service Relations** is a property for the Feature in the DataPoint Type of Service Type that is defined in the System stage. So If a system has a Service like “service1”, it has a Service Type and this Service Type has a DataPoint Type. In this DataPoint Type there are features that is requires some amount of features. In this **Service Relations**, user must choose which services provide that feature. It’s not obligatory, so it still matches the System constraint even if there is no relation though it is needed.

**Parameter Values** are the variables that was defined in the Service Type stage also if there is parameters that is not defined in the Service Type stage it must define in this stage. All parameters that wasn’t assigned any value in the DataPoint Type and Service Type, must be defined in the System stage

Lastly, there is a stage for creating the System Bundle and this will be told later in the ***“Creating a Bundle”*** section.

### Web based code editing

For Every Service Type, they have codes that is composed in the back end side. Some portion of this code cannot be changed and some can be changed. We call the second part as “TODO” part. In the Service Type Edit mode, user can code for TODO parts and can save these TODO parts in the database separately  
  


# Back end

### MongoDB access

We use MongoDB for storing the data. It’s NoSQL database and it is not like a RDBMS. There are only relations between entities. It’s not relational and there are no tables and schemas. Data is stored as JSON documents, fields can vary from document to document. With MongoDB Compass, we created a database that is named “gamma” and for testing purposes we created a second database that is named “test”. MongoDB Compass is a tool for MongoDB. You can do CRUD operations or manage your database with this tool easily.

To access the database we have an access config and when we run the back end application, first we initialize the database and collections.

### API development

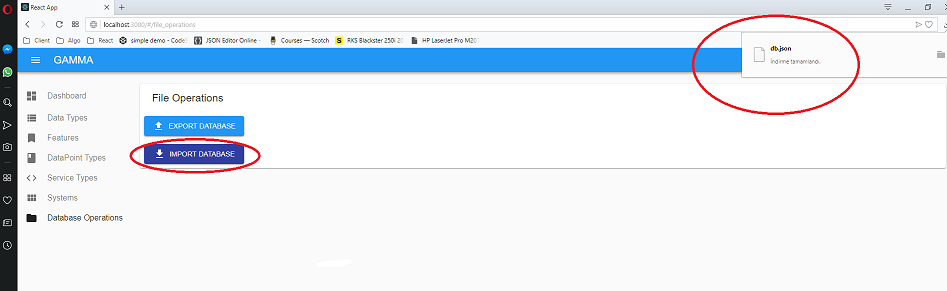
### Code generation

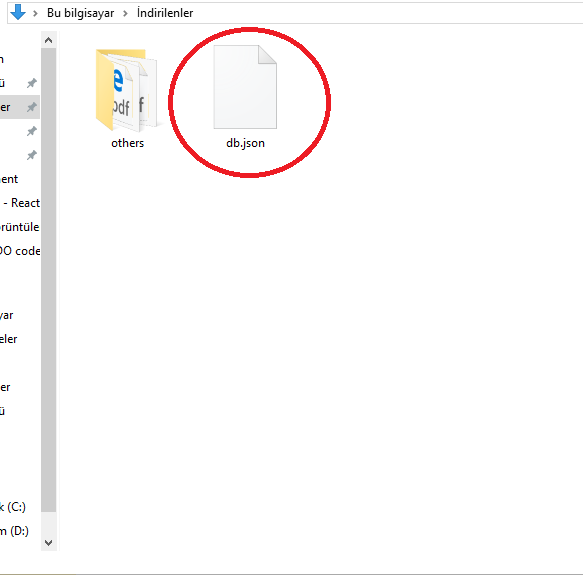
To generate Service Type Code template and app.js from System page, we use EJS templates. It is an Embedded JavaScript templating language lets you generate HTML markup with plain JavaScript.

Services in the created system will be automatically constructed and initialized in the app.js by the ejs template. They are implemented based on the given inputs at the front end. However zwave and zigbee communicator services are special and they do not generated automatically. Communicator services are implemented hardcoded in the EJS template and existed in all generated system bundles.

### DB import/export functions

With the DB Export function, users can download all the entities that is saved in the database so far to their local computer as a .json file





With the DB Import function, users can upload a .json file to the database. But this process will overwrite all old entities and so deletes the old entities.

# Demonstration

### Gerade office

### Iotiq office

### 

# Device services

Device services are interfaces of the devices in the system. They extends the same service template(BaseSt) and each one is implemented with different features and functionalities. They are finalized at the service type level. Assigned functionalities and relations of the devices are controlled and run by device services. They receives messages through communicator services and responds as decided formerly.

### Door sensors

Door sensor service can be interface of either Xiaomi door sensor or Fibaro door sensor. Xiaomi door sensor service runs with **zigbee communicator** service and Fibaro door sensor runs with **zwave communicator** service.

Door sensor service has **Binary sensor** feature which comes from door sensor datapoint type. This feature indicates on and off states of the door sensor device. When its state is changed, it receives new value through communicator service and set its new value. When new value assigned door sensor service inform the other services which has relation with it.

Door sensor devices stay in sleeping mode and only wake up shortly when state changed or button on them is pressed. Also door sensors wake up in regular intervals. To receive message other than **feature value changed** message or send message, user needs to wake it up or wait for its regular wake up. Because, devices send/receive signals when they are awake.

### Temperature/Humidity/Pressure sensors

Temperature/Humidity/Pressure sensor service is the interface of Temperature/Humidity/Pressure device. It runs with **zigbee communicator** service.

Temperature/Humidity/Pressure service has three features: **Temperature**, **Humidity**, **Pressure**. It gets value for each feature and make different decisions for each one. Basic decision of the functions is setting new value and informing the services that has relation with it. User can improve the decision function when implementing service at the service type level.

Temperature/Humidity/Pressure sensors stay in sleeping mode and only wake up in shortly for send new feature value and when the button on them is pressed. Also they wake up at regular intervals. To complete a message interaction with the device, it should be waked up.

### Blind controller

Blind controller service is the interface of the BlauPunkt blind control device.It runs with **zigbee communicator** service.

Blind controller service has **Binary sensor**, **Binary switch**, **Level sensor**, **Direction sensor** features. When it receives a message through **zigbee communicator** it analyze the message and understand which feature value the message has. It will execute different responses for different features and values. Basic response for value change is setting new value and informing the services which has relation with it. User can implement more complex responses while generating code at the service type level.

### Wall plug

Wall plug service is the interface of the Fibaro wall plug device. It runs with **zwave communicator** service.

Wall plug service has **Binary switch** feature and it has two states: on and off. When it receives message through **zwave communicator** service, it checks whether it is new feature value message. When a new binary switch value is received, it sets the new value to the binary switch feature and informs the services which has relation with it.

Wall plug devices are stays in awake mode always. They plugged in power sockets and use electricity of the house directly. Therefore they can receive message any time. However they are programmed to send **feature value changed** message while running. To receive other messages user needs to physical interaction such as pressing the button on it.

### Orchestrator service (binary coupler)

Orchestrator service is a special service because it does not has a device. It is a virtual device service. It does not run with any communicator service but it builds relations with other services.

Role of the orchestrator service is to create a bridge between two device services. It has “Binary switch” and “Binary sensor” features which are the same features of the devices it has relation. Unlike other device services it receives feature value messages from other services. When other services informs it for a change at feature value it sets the new value and inform the services it has relation. Briefly, it transports new feature value from one device service to another.

Since orchestrator service is a virtual device service it does not have sleeping or awake states. It receives message as soon as the message of feature value change is sent. Also it does not have any requirement for sending message so it sends message as soon as finished process.

# Communication services

Communicator services are interfaces of the dongle devices in the system. They are controllers of their local networks. Device messages are received by communicator services firstly. Then communicator services analyze the message and transmit it to its destination. Also messages are sent to the devices through communicator services.

### Zwave

Zwave communicator service is the interface of the zwave dongle device. It coordinator of the zwave network and controller of the devices use zwave wireless protocol. Zwave communicator uses **openzwave-shared** library therefore the basic functionalities are predefined. User needs to implement more complex functions by using predefined library features and functions.

When user run the **zwave communicator service,** it starts the dongle, then dongle scans the devices in the network. After scan completed dongle sends the **scan complete** message and enters the listening mode for receiving device messages. When a device communicates with zwave protocol sends a message, zwave communication service receive the message and if the message type has a response function it runs the response function.

### Zigbee

Zigbee controller is the interface of the zigbee dongle device. It is coordinator of the zigbee network and the controller of the devices communicates with zigbee wireless protocol. We implemented a zigbee communication library and **zigbee communicator service** uses this library. The library has predefined basic functionalities of zigbee communication. User needs to implement more complex functions by using basic functions of the library.

**Zigbee communicator service** runs the dongle and make it join or create a network. Then it adjust the security and join permissions . After finished initialization **zigbee communicator service** listens the port and waits device messages. When a device message received it analyzes the message, then transmits to the correct device service. Similarly it takes messages from device services and sends them to the devices.

Basic functionality of the **zigbee communicator service** is receive and send data events. User should implement the other functions at the service type level code implementation.

# Deployment

### Creating a bundle

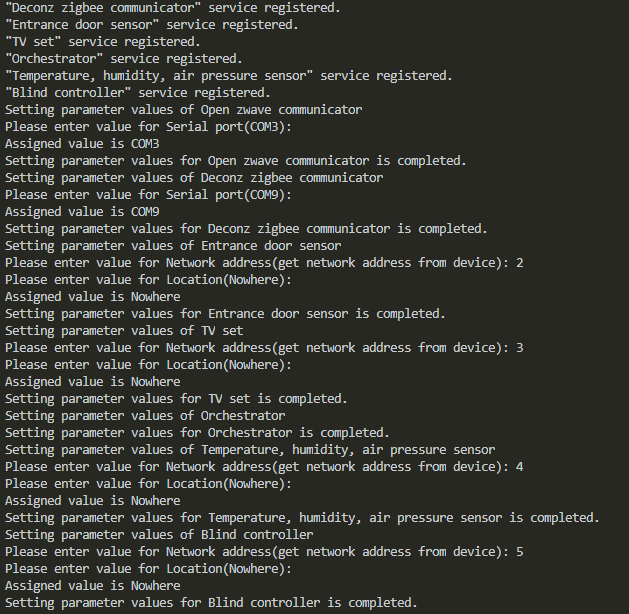
Creating bundle stage is done in the System page. There is a button for that and after pressing that button in a temporary directory in the backend, there will be generated an app.js file and service type codes that is added to the system. In that folder, there are some static files too. After generating app.js and service type code, a zip file of this temporary folder will be downloaded to the local. If the user unzip the file and runs the app.js, system must start to run successfully.

### System deployment

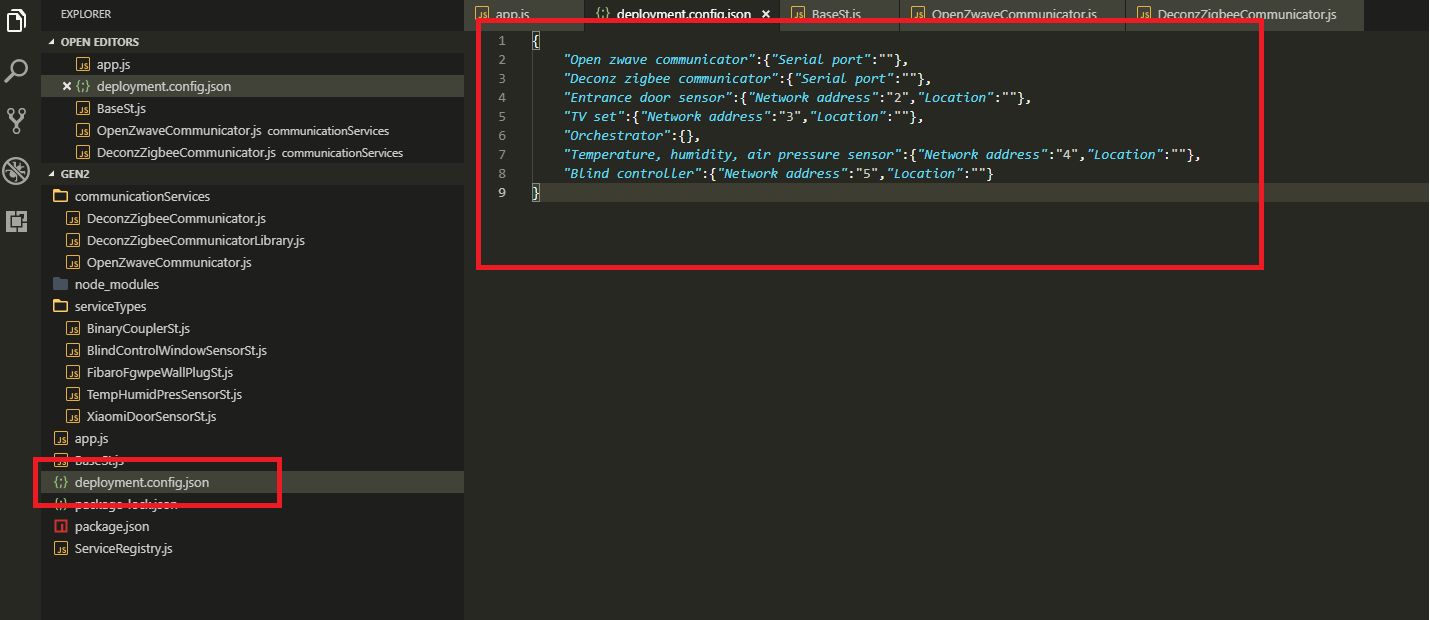
There are two options for running the system bundle. First one is running without any parameter( npm start). If system bundle is started without parameter, it initialize the services with the parameters that are read from a local file. Then it starts the services and run the system automatically. The other option is running the system bundle with -pair parameter( npm start -pair). When system bundle runs with -pair parameter it initializes the services and then starts device pairing process. Device pairing process will be needed either at the first installation of the system or when device ids are changed and going to be updated.

System takes the parameter values entered at the service level and system level as default values. For each service, system asks user a value or set as default. User uses command line to enter a value for parameter. Other than network address, all parameters are treated same. Network address is differentiated because it is not asked for communicator services and its default value is retrieving the id from the device.

To retrieve the device id, communicator service asks user to take some physical action on the device and then enter the listening mode. It collects the messages for ten second intervals and if there are only one device that sent message in a interval, then it takes the id of the device from the message. Communicator service reenter ten seconds intervals of collecting messages until it identify a device or running system terminated.



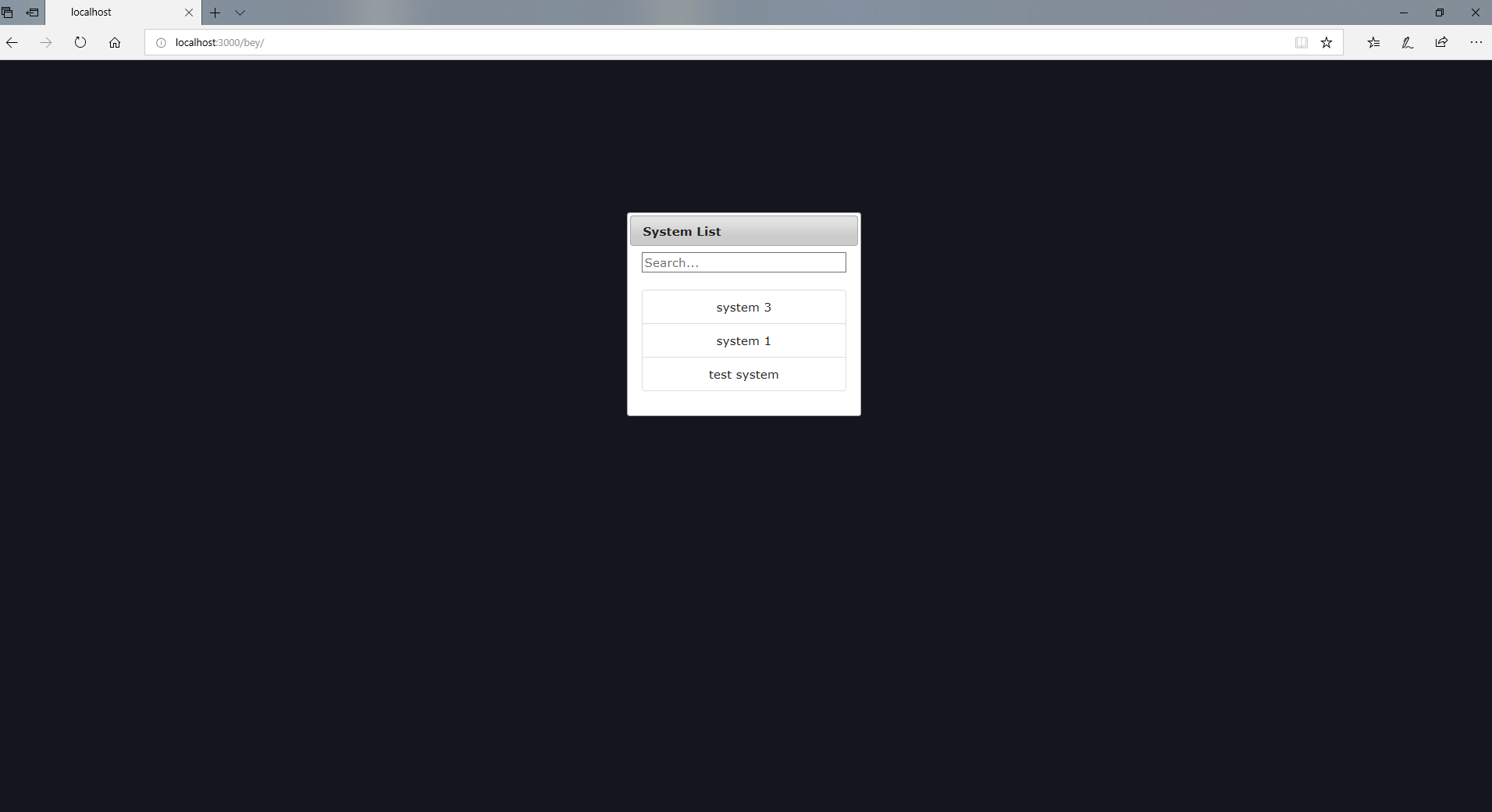
After system bundle main function gets all the parameter values and updated them, it writes parameter values into a local file as a json object.



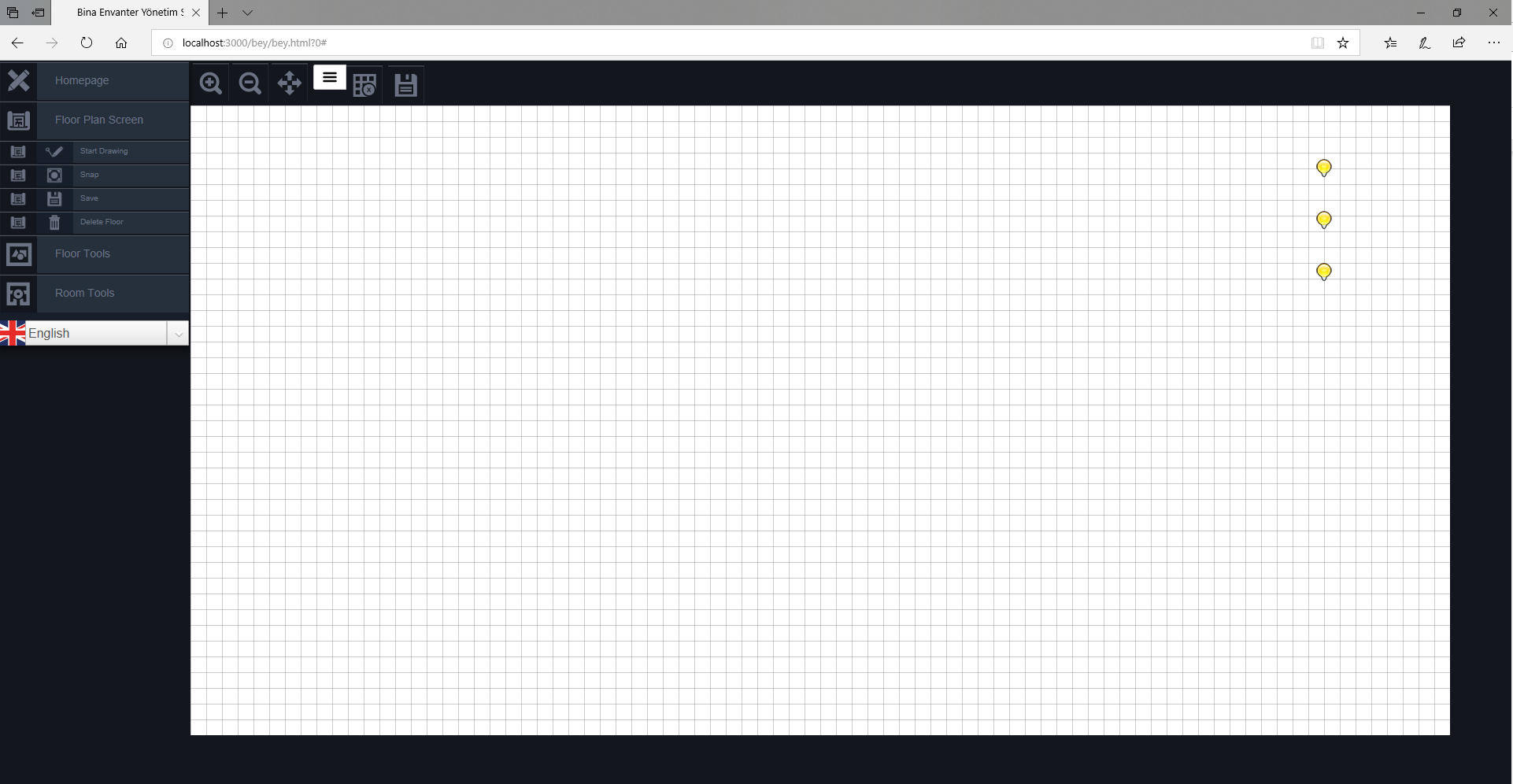
# Operation

### Floor design

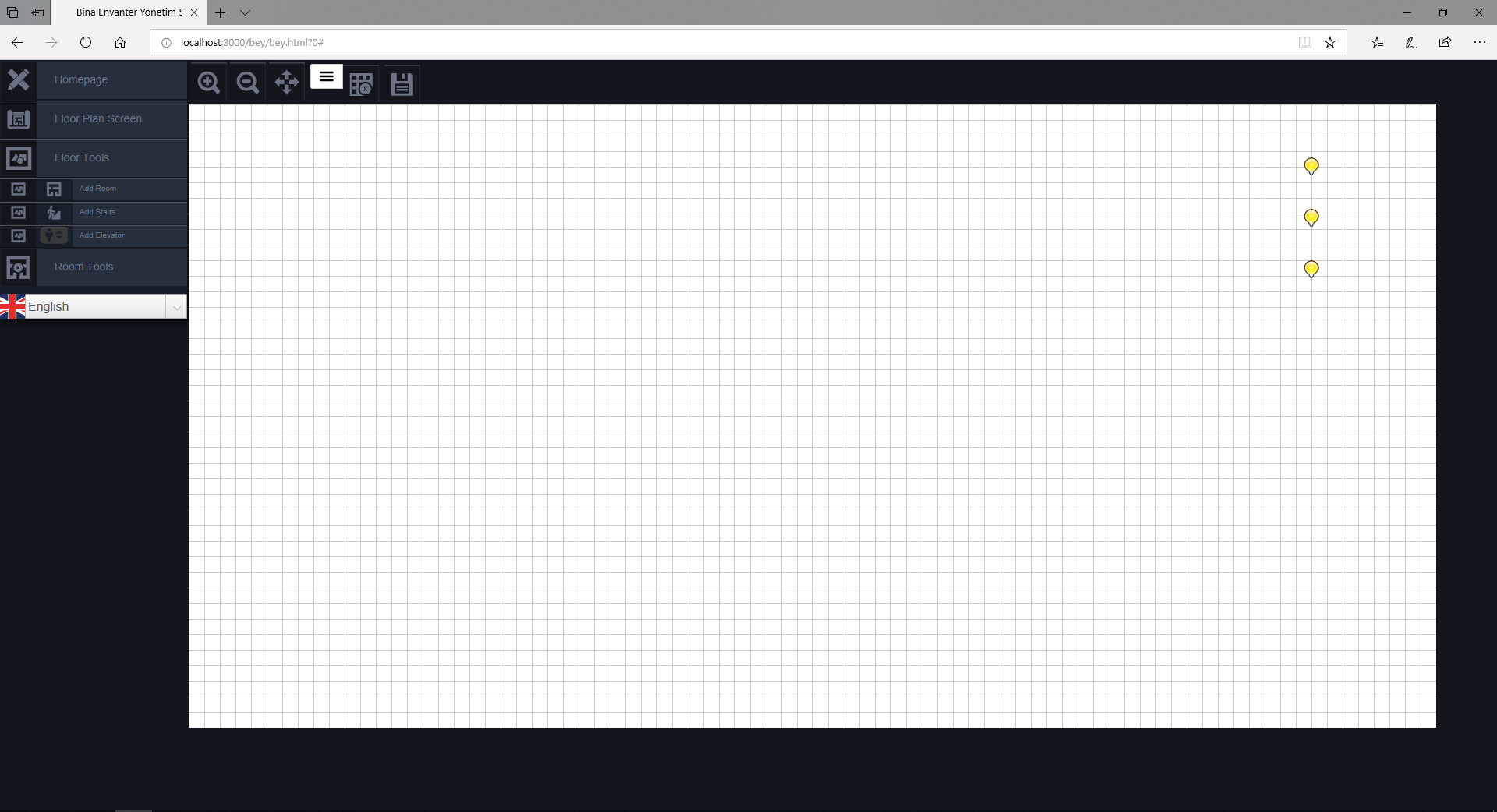
GAMMA uses BEY visualization tool for floor design. BEY tool is modified and integrated into the GAMMA system. User goes to BEY tool via button on the GAMMA front end. It lists systems created at the front end. User can choose a system on the list or filter the list from the search bar. It filters the systems by showing only systems that contains the searched word in themselves.

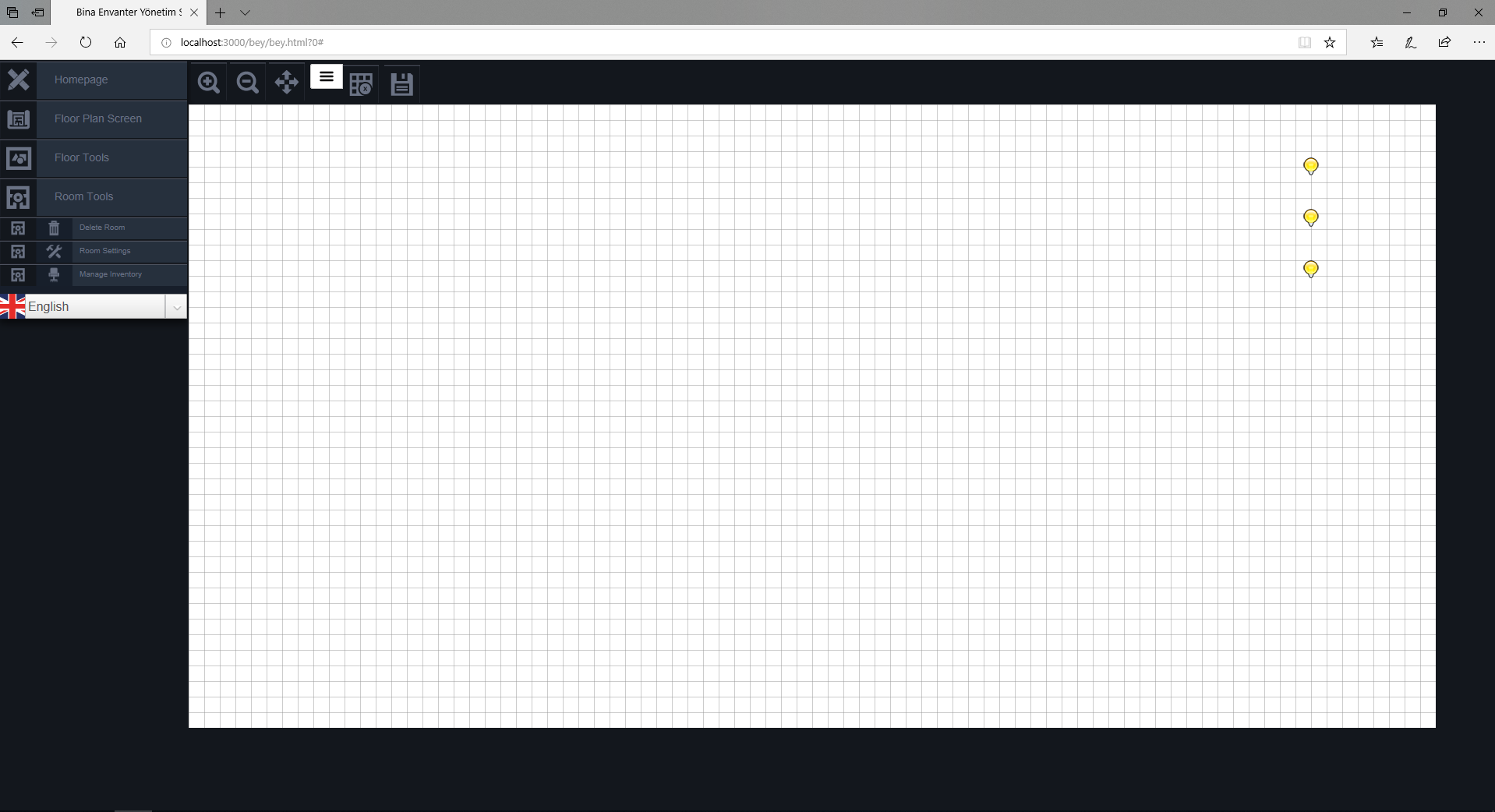


Each floor is assigned to a system and by choosing a system, user is directed to floor design page. BEY loads floor data from mongo database, and save edits to the same database. At the initial load since there is not any floor design yet, user sees services at the right side of the canvas ordered top to bottom. Device and Communicator services are both showed as the same type services.

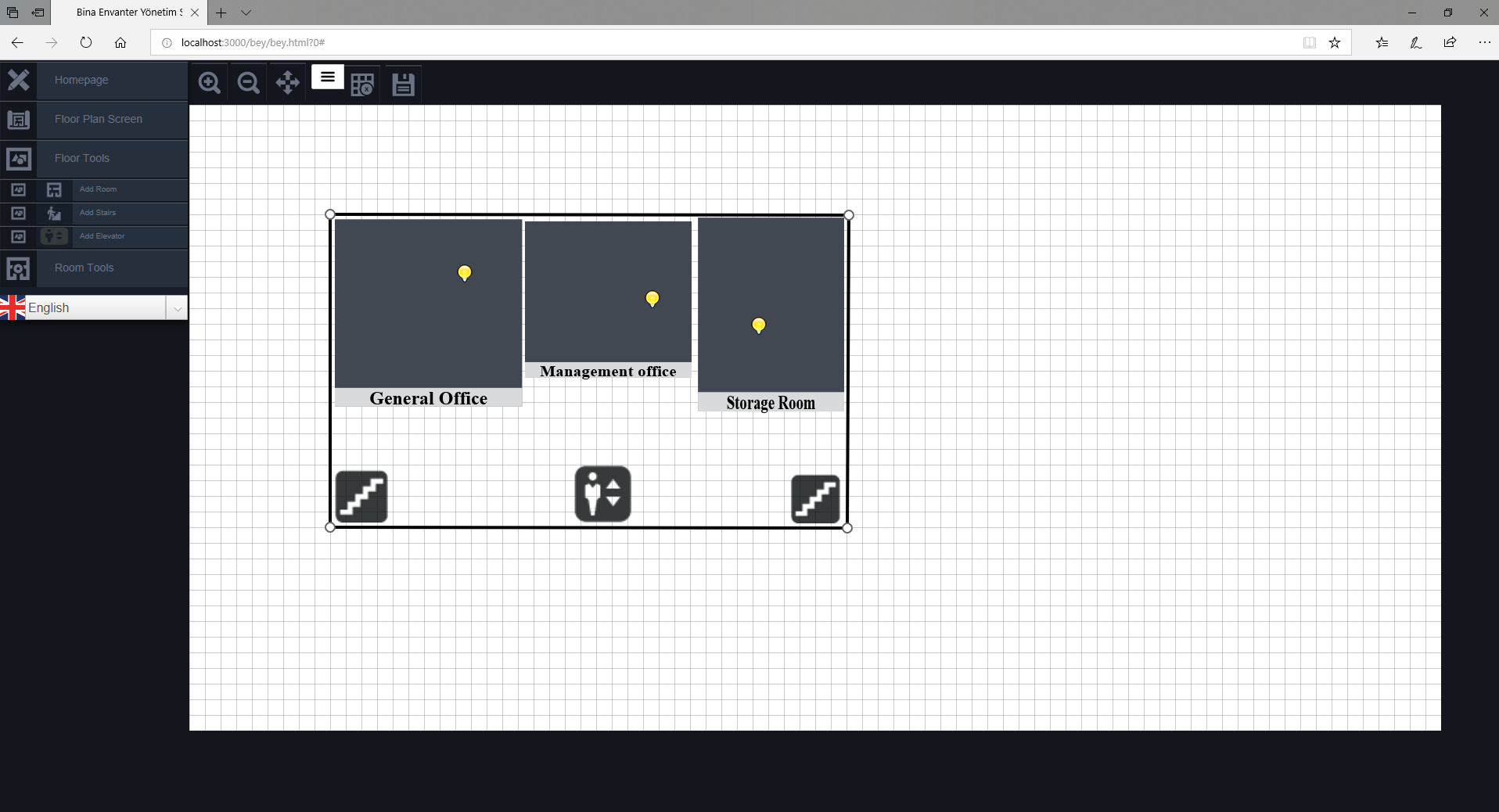


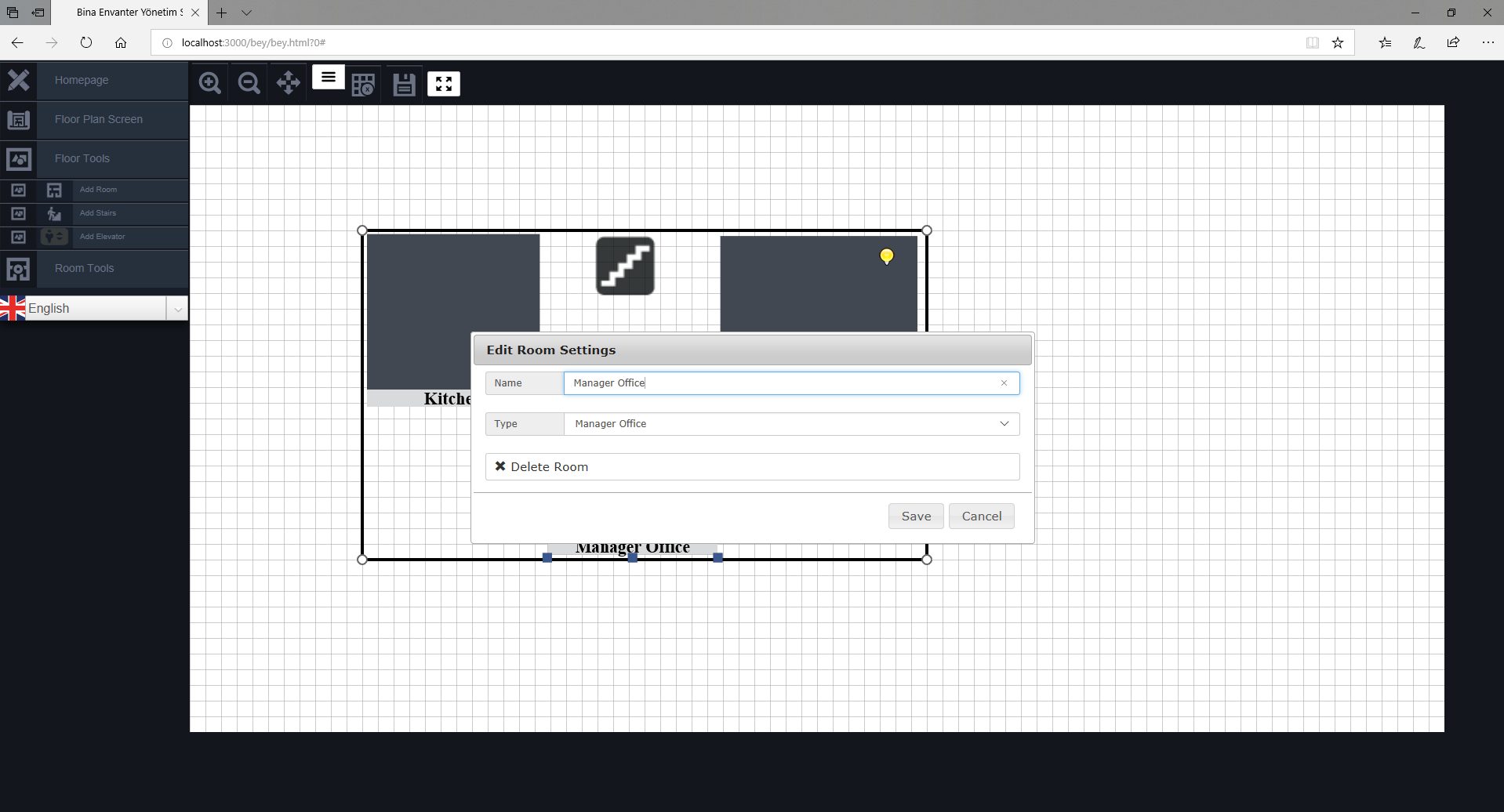
On the left side of the page there is a menu that allows user to navigate back to GAMMA front end, choose language and use functionality of the BEY tool. Under “Floor Plan Screen” header, user can draw floor lines which indicates floor borders and room walls. After lines are drew, snap option makes lines straighter. User can also save and delete floor under this header. Under “Floor Tools” header, user can add floor objects such as rooms, stairs, elevators. It added new floor objects left-top of the canvas, then user can drag them on the canvas. Under “Room Tools” header, user can delete floor objects, edit their settings, and manage their inventories.

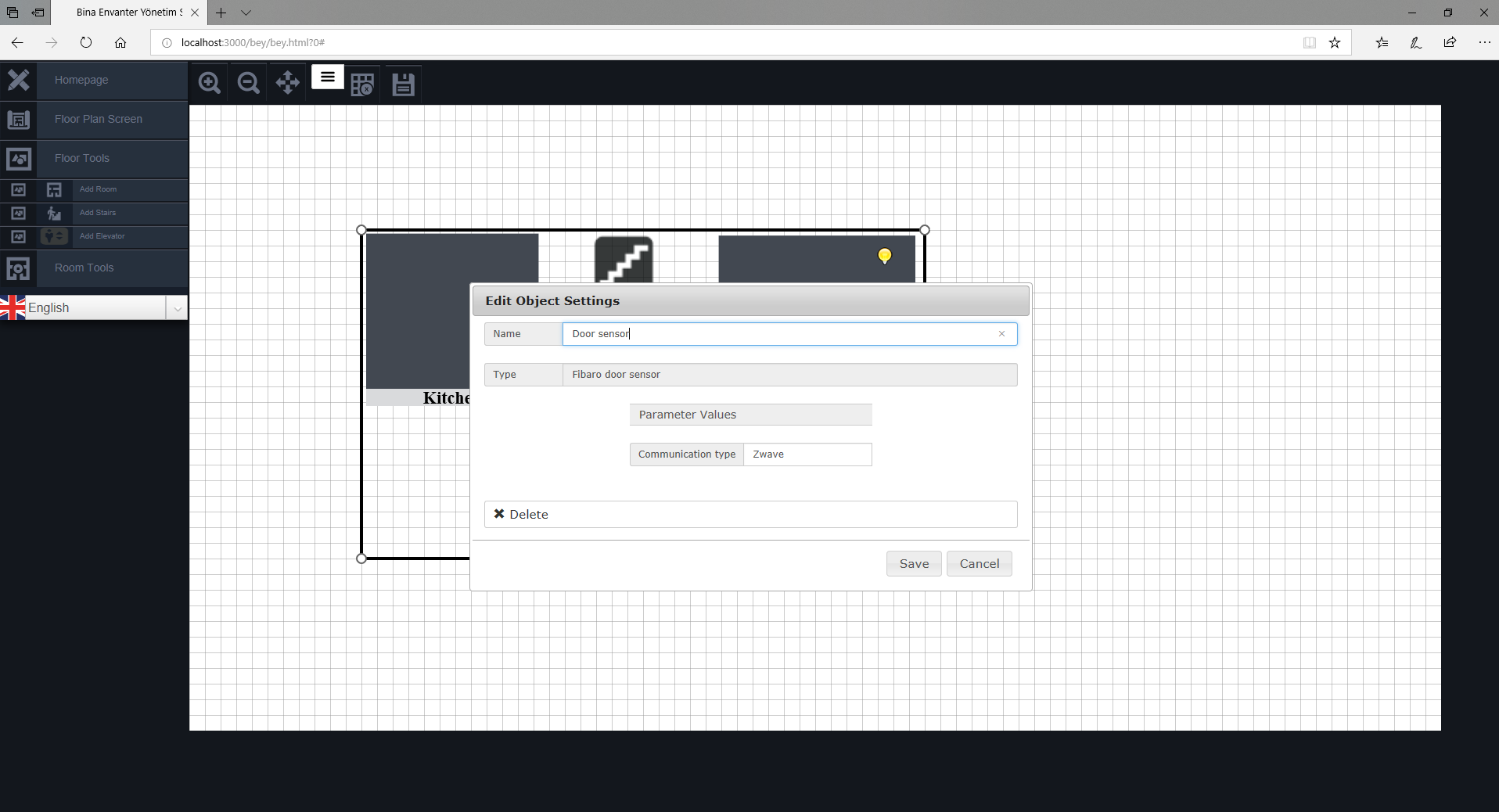




Devices are attached to floor objects by dragging them on the objects. Devices dragged on a room are added into the inventory of that room and move along with it. Rooms and devices has setting pages. On the device settings page user can edit name and parameter values of the device, and see service type of the device but cannot edit the service type. The saved changes at the device settings page are updated on the database too. User can also remove the service from the system by deleting the device. To remove the device, system asks a confirmation from user. On the room settings page name and type of the room can be seen and updated. Also user can delete room from its settings page. Same as the device deletion, user gets a confirmation message.







On the top of the page there is an button menu that has buttons that are not in the general menu of the page. User can zoom in and zoom out with the corresponding buttons. “Selectall” button select all the objects on the canvas and move them together. There are shortcut buttons for settings and save. This settings button will open the room settings page for room objects and device settings page for devices. Save button save the floor into the database. “Togglegrid” button change the background between checker and white. Lastly when user clicks on a floor object, resize button will appear. When click on resize button resizing points of the floor object will shown and object can be resized.

