CONTENTS

INTRODUCTION

You have to develop a web service for managing the modular automation system “IoT Manager” based on ESP32/ESP8266 microcontrollers.

The automation system is a kind of application in Python and C ++, the source code of which will be provided to you in media files. You do NOT need to interact with this application, compile, run, or work with Python and C++. You need to work with the project structure. This project has the following structure:

```
data_svelte
include
lib
src/
  classes/
  modules/
    exec/
      moduleA
      moduleB
    sensors/
      moduleC
      moduleD
    virtual/
      moduleE
      moduleF
  utils/
tools
training
.gitignore
LICENSE
PrepareProject.py
README.md
myProfile.json
platformio.ini
```

This system contains a number of modules that are located in the **src/modules** folder. Modules are grouped by their type, for example, executable (exec), sensors (sensors), virtual (virtual). Inside the group are folders with modules - a folder can have any name and it can contain a set of files and folders.

At the root of the application is the **myProfile.json** file, which contains the application's settings, as well as a list of modules.

The customer wants the future support of the developed service to be simple, so he insists on using one of the server-side frameworks such as Laravel or Yii.

Example myProfile.json file:

```
{
    "iotmSettings": {
       "settings": "",
       "name": "IoTmanagerVer4",
       "apssid": "IoTmanager",
       "appass": "",
       "routerssid": "rise",
       "routerpass": "hostel3333",
       "timezone": 1,
       "ntp": "pool.ntp.org",
       "weblogin": "admin",
       "webpass": "admin",
       "mqttServer": "m2.wqtt.ru",
       "mqttPort": 8021,
       "mqttPrefix": "/risenew",
       "mqttUser": "rise",
       "mqttPass": "3hostel3",
       "serverip": "http://iotmanager.org",
       "log": 0,
       "mqttin": 0
    },
    "projectProp": {
       "platformio": {
           "default_envs": "esp8266_4mb",
           "data_dir": "data_svelte"
       }
    },
    "modules": {
        "virtual": [
            { "path": "src\\modules\\virtual\\Logging", "active": true },
            { "path": "src\\modules\\virtual\\Timer", "active": true }
        ],
        "sensors": [
            { "path": "src\\modules\\sensors\\Ads1115", "active": false },
            { "path": "src\\modules\\sensors\\Aht20", "active": true }
        ],
        "exec": [
            { "path": "src\\modules\\exec\\ButtonIn", "active": true },
            { "path": "src\\modules\\exec\\ButtonOut", "active": true }
        ],
        "display": [
            { "path": "src\\modules\\display\\Lcd2004", "active": true }
        ]
    }
}
```

This file has a number of settings, properties and a list of modules that are available in the system. Each module in this file has a path to the folder with the module and an active property, which is responsible for whether this module is included in the assembly or not.

The service you develop should also have its own API that allows external applications to get a list of your projects and view information about the project.

DESCRIPTION OF PROJECT AND TASKS

Your task is to develop a web service that allows users to create assemblies of this system that they need, but with a different set of modules. Assemblies do not need to be compiled, run, or interacted in any way with C++ and Python. All you need to do is create a zip archive for the project containing a different set of modules and a valid myProfile.json file.

Work algorithm:

1. The source files of "IoT Manager" are located on the server, in the «**iot**» directory, in the root of the module.
2. Service user visits your service
3. The user goes to the page with modules
4. User uploads a zip archive containing at least 1 module folder
5. The user goes to the project creation page
6. The user specifies the required settings, selects which modules of the source system (located in the iot directory) should be included in the project, selects which of the native modules (loaded earlier) are needed and creates the project.
7. Information about the project is stored in the database, the user gets the opportunity to download the zip-archive with the "IoT Manager" system, which contains only those modules that were specified. The myProfile.json file contains the correct information about the modules included in the assembly and the system settings.

You need to develop a web service with the following structure:

- Web interface:
    - Registration
    - Login
    - Modules Page
    - Add module page

        - Projects Page
        - Project creation page
        - Access keys page
- API:
        - Getting a list of projects

Interface part

Registration page

Anyone should be able to register in the system using an email address and a password.

Login page

Registered users must be able to log in using the combination of email and password provided during registration.

After a successful login, the user should see a page with projects.

Page with projects

On this page, users should see their previously created projects.

Each project must contain:

- Name of the project
- Delete button
- Button for downloading the zip-archive with the project

Modules page

This page should display the modules that the user has uploaded.

Each module should display the following information:

- Module name - taken from the modinfo.json file located in the folder with the module
- Download button - when clicked, a zip-archive with this module should be downloaded
- Delete button - when pressed, this module should be removed from the system

This page should also contain a button to go to the page for adding a new module.

Add Modules Page

This page should contain a form for uploading modules. The form should consist of 1 field for selecting a zip archive and a button for submitting the form.

The form data must be validated on the server side: the module file must be a mandatory zip archive.

As a result of submitting the form, the modules contained in the uploaded archive should be added to the system and displayed on the modules page of the user who uploaded them.

It is assumed that the zip archive always contains at least 1 module folder. If there are 3 folders in the zip archive, then 3 modules must be added to the system.

Inside each module folder, there must be a modinfo.json file. The system takes the name of the module from it.

Project creation page

On this page, you need to display a form containing the following fields:

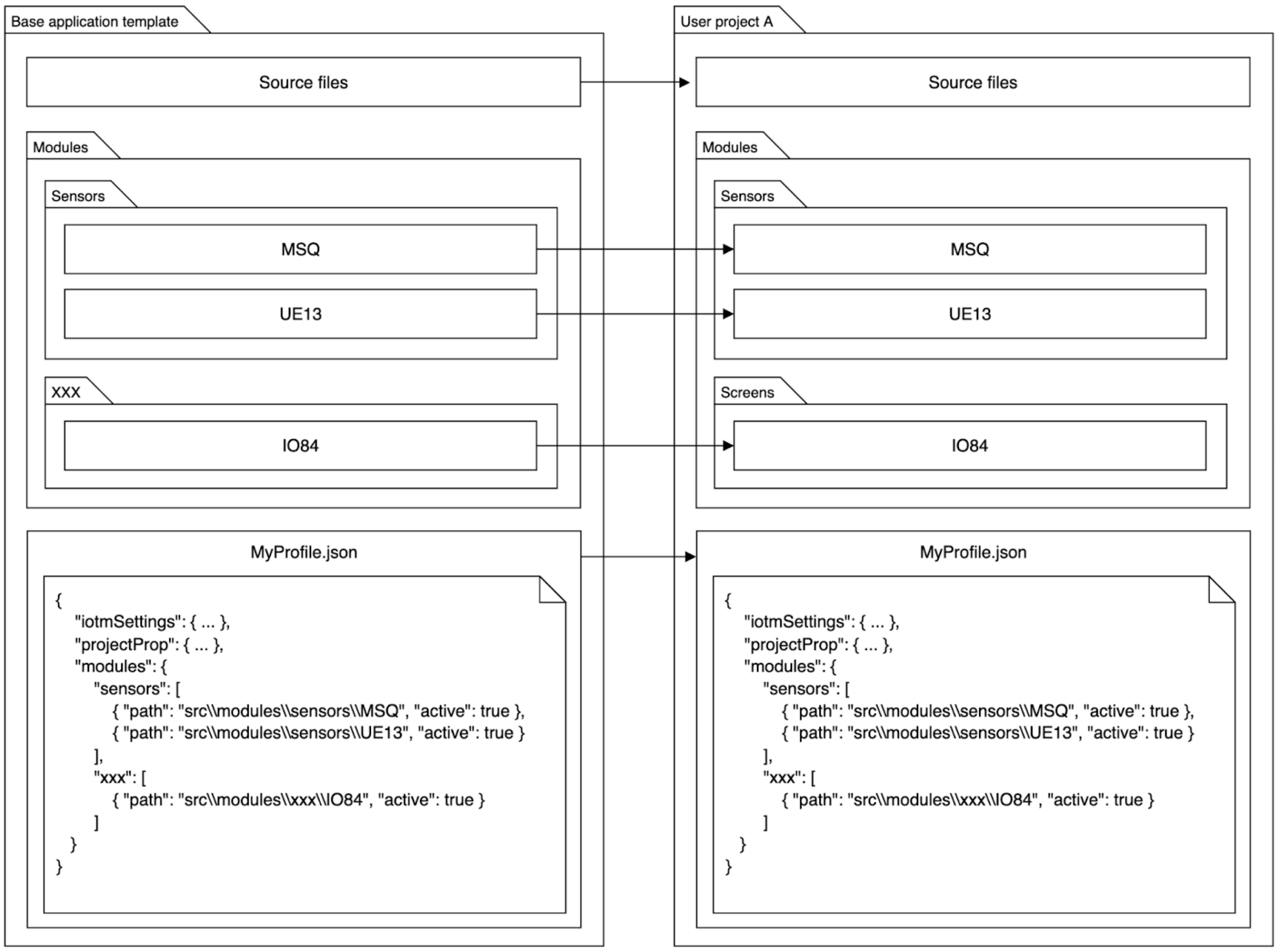
- The name of the project being created is required
- Login from Wi-Fi – required field, corresponds to the routerssid field in the myProfile.json file
- Wi-Fi password - required field, corresponds to the routerpass field in the myProfile.json file
- List of modules from the base template - 0 or more modules
- List of custom modules (uploaded to your account) - 0 or more modules
- Send button

By default, all modules in the base project must be checked.

In the process of filling out the form, the user can select only those modules of the base template that he needs, as well as his own modules, which should also be included in the assembly.

The base application template source modules must be taken from the base template folder located in the "iot" directory at the module root. If you replace the original template files with others manually, then everything should work and those modules that are currently in the template folder should be displayed.

The user can mark all the modules of the base template and not add his own, then the created project will be exactly the same as the base project template (Figure 1).

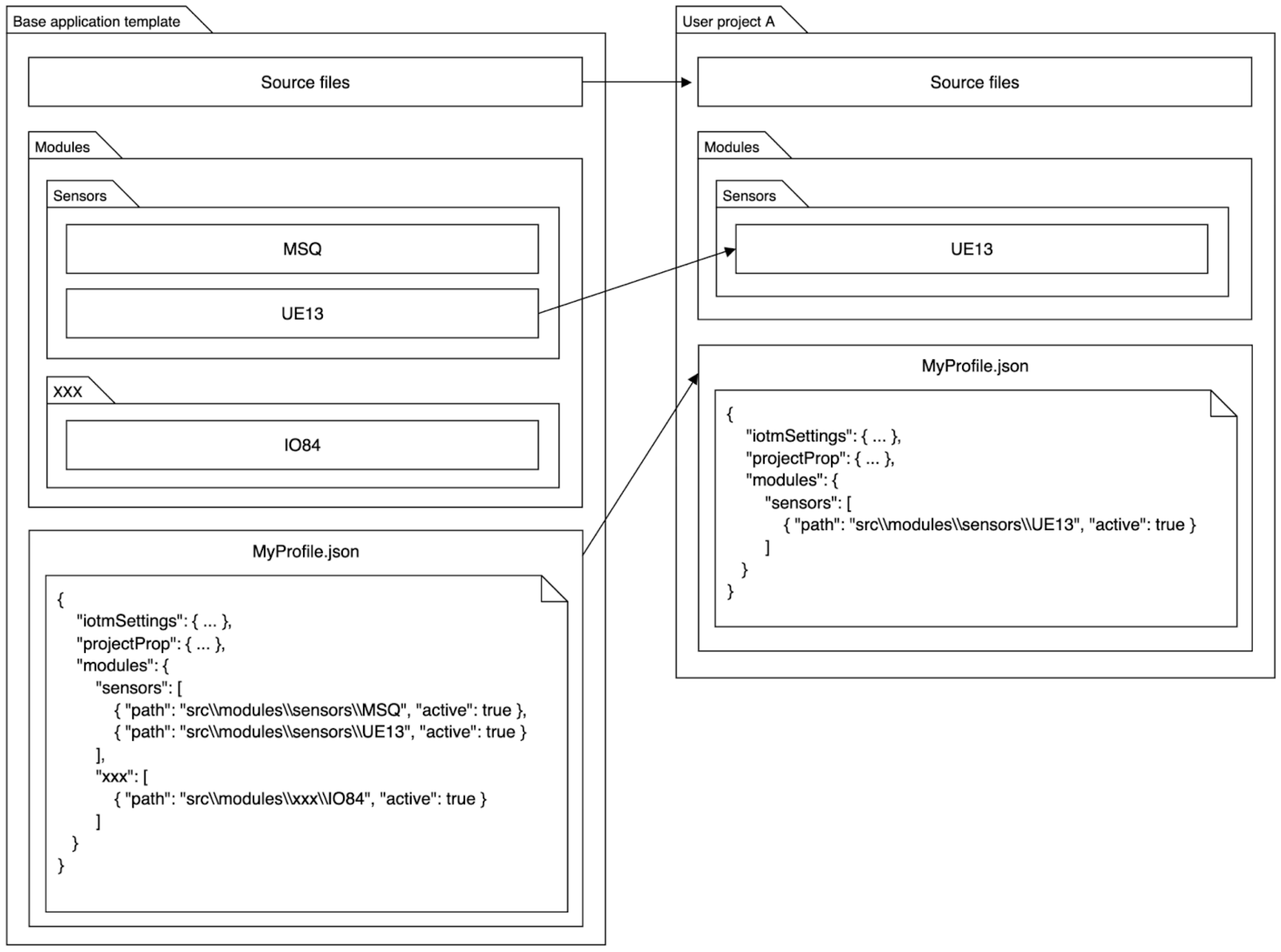


```
{
    "iotmSettings": { ... },
    "projectProp": { ... },
    "modules": {
        "sensors": [
            { "path": "src\\modules\\sensors\\MSQ", "active": true },
            { "path": "src\\modules\\sensors\\UE13", "active": true }
        ],
        "xxx": [
            { "path": "src\\modules\\xxx\\IO84", "active": true }
        ]
    }
}
```

```
{
    "iotmSettings": { ... },
    "projectProp": { ... },
    "modules": {
        "sensors": [
            { "path": "src\\modules\\sensors\\MSQ", "active": true },
            { "path": "src\\modules\\sensors\\UE13", "active": true }
        ],
        "xxx": [
            { "path": "src\\modules\\xxx\\IO84", "active": true }
        ]
    }
}
```

Figure 1 - Building a project based on a complete base template

If the user marks only a part of the modules of the base project, then the unmarked modules should not be included in the assembly (Figure 2).

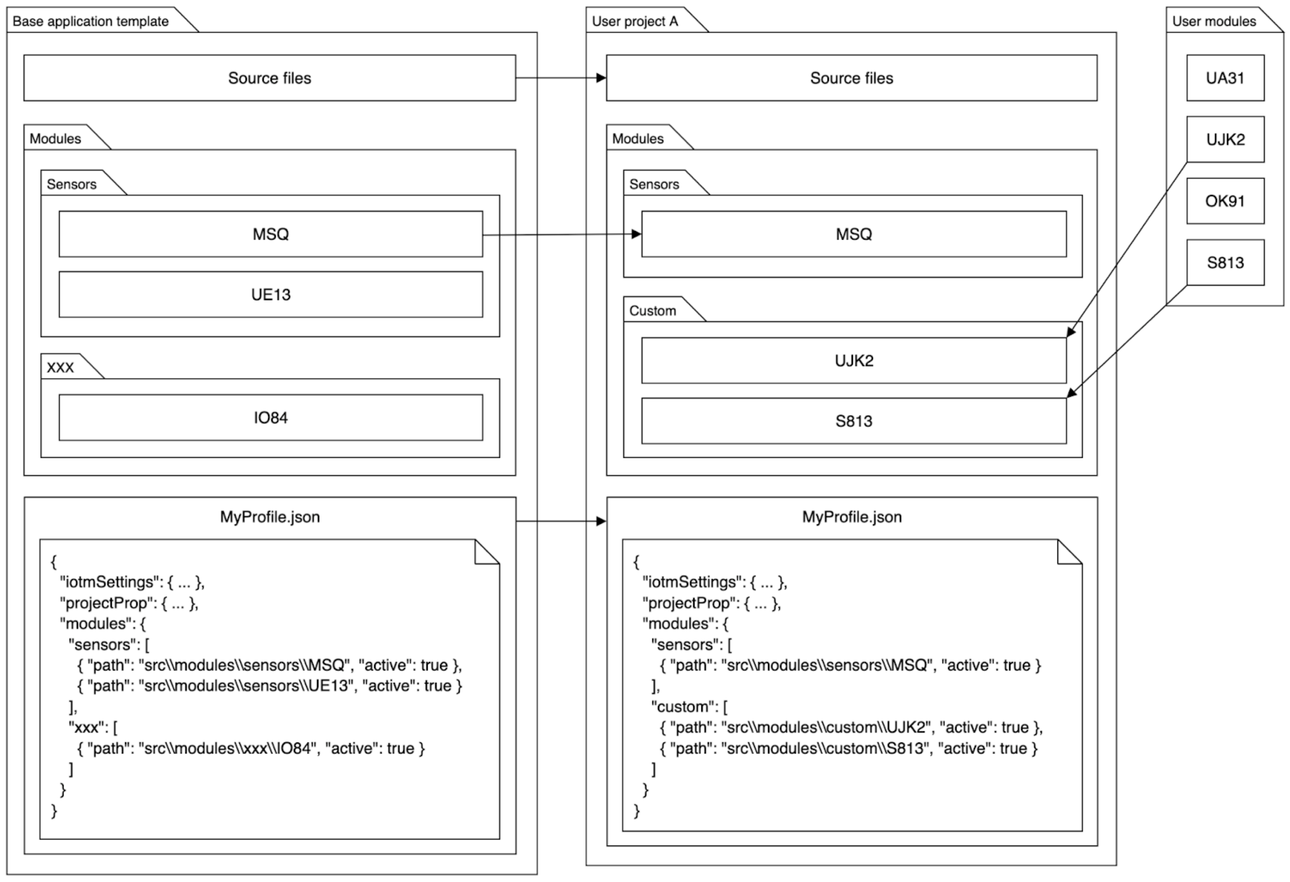


```
{
    "iotmSettings": { ... },
    "projectProp": { ... },
    "modules": {
        "sensors": [
            { "path": "src\\modules\\sensors\\MSQ", "active": true },
            { "path": "src\\modules\\sensors\\UE13", "active": true }
        ],
        "xxx": [
            { "path": "src\\modules\\xxx\\IO84", "active": true }
        ]
    }
}
```

```
{
    "iotmSettings": { ... },
    "projectProp": { ... },
    "modules": {
        "sensors": [
            { "path": "src\\modules\\sensors\\UE13", "active": true }
        ]
    }
}
```

Figure 2 - Project assembly based on part of the base template

Also, the user can select his modules too, then they must also be included in the assembly. Custom modules should go into the src/modules/custom folder (Figure 3).



```
{
  "iotmSettings": { ... },
  "projectProp": { ... },
  "modules": {
    "sensors": [
      { "path": "src\\modules\\sensors\\MSQ", "active": true },
      { "path": "src\\modules\\sensors\\UE13", "active": true }
    ],
    "xxx": [
      { "path": "src\\modules\\xxx\\IO84", "active": true }
    ]
  }
}
```

```
{
  "iotmSettings": { ... },
  "projectProp": { ... },
  "modules": {
    "sensors": [
      { "path": "src\\modules\\sensors\\MSQ", "active": true }
    ],
    "custom": [
      { "path": "src\\modules\\custom\\UJK2", "active": true },
      { "path": "src\\modules\\custom\\S813", "active": true }
    ]
  }
}
```

Figure 3 - Project assembly based on part of the base template and custom modules

After submitting the form, a zip archive should be created containing all the selected modules, as well as a valid myProject.json file. The user should be redirected to the page with their projects.

The myProject.json file is formed on the basis of a ready-made file from the base template myProfile.json with the replacement of the specified data on the project creation page.

Also, after creating the project, you must send a request from your server to the provided API to inform about the successful creation of a new assembly.

External API to report successful build

URL: `http://nvafmzc-m2.wsr.ru/api/builds`
Method: `POST`
Request Headers:

- `Accept: application/json`
- `Content-Type: application/json`

Request body:

```
{
    “competitor”: <your static name>,
    “project”: “Project name”,
    “url”: “http://<competitor_login>-m2.wsr.ru/files/<unique_filename>.zip”
}
```

Response status: `204`

Access keys

On this page, you need to display information (name and token) about user access keys that will allow you to interact with your API. An access-key is simply a named API token that the application will use to access the system.

Each key on the page should have a delete button.

The page should have a form for adding a new key. To add, just enter the name of the key and submit the form.

API

Request without access key

When attempting to make any API request without a valid token, the following response should be returned to the client:

Response Headers:

- `Content-Type: application/json`

Response status: 401

Response body:

```
{
   code: 401,
   message: "Authorization error",
   error: "AUTH_TOKEN_INCORRECT",
   errors: []
}
```

Getting a list of user projects

URL: /api/projects

Method: GET

Request Headers:

- `Accept: application/json`
- `Authorization: Bearer <token>`

Response status: 200

Response body:

```
{
   data: [
       {
           "id": 1,
           "name": "Project A",
           "file": "http://<competitor_login>-m2.wsr.ru/files/<unique_filename>.zip"
       }
   ]
}
```

The "file" field should contain a link to get a zip file with the project archive.

You need to protect the file from viewing and downloading by using the application token API.

INSTRUCTIONS FOR THE COMPETITOR

Save your work on the server in the second module folder.

The service you developed should open at http://xxx-m2.wsr.ru, where xxx is your username.

The API you developed should respond to http://xxx-m2.wsr.ru/api, where xxx is your username.

Jobs not saved to the server, or jobs that were saved in error or deployed incorrectly, will not be reviewed. So make sure your service is up and running.

The web interface will be evaluated in the Google Chrome browser and API will be evaluated using Postman.

MARKING SCHEME

CRITERION	MEAS. MARKS	JUDG. MARKS	TOTAL
Layout	6.00	0.00	6.00
Functional	8.00	0.00	8.00
Project creation	5.30	0.00	5.30
Access keys	1.00	0.00	1.00
API	2.70	0.00	2.70
General	2.00	1.00	3.00
Code quality	0.00	3.00	3.00
Design	0.00	6.00	6.00
TOTAL	**25.00**	**10.00**	**35.00**