Vortex WiFi Valve Software Architecture

Mobile App And WebServer Communication (REST / WebSocket)

User Authentication & Device Data Flow

1. User Credential Provisioning

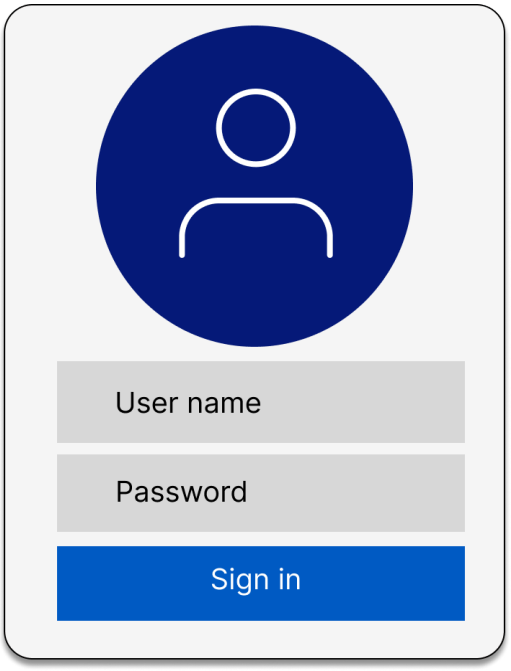
- The **Vortex Network Admin** creates a **username** and **password**.
- Credentials are sent to the user **via email**.
- The mobile app uses these credentials for login.

2. Mobile App Login (REST API)

Login Request

- When the user enters their username and password and taps **Sign In**, the mobile app sends a REST post request to **login.php** :

```
{
  "username": "testuser",
  "password": "mypassword123"
}
```



Server Response (Successful Authentication)

- If the credentials are valid, the server responds with:

```
{
  "success": true,
  "message": "Login successful",
  "access_token": "<jwt_token_here>",
  "refresh_token": "no_refresh",
  "user": {
    "id": "<user_id>",
    "name": "<user_name>",
    "email": "<user_email>",
    "contact": null
  }
}
```

- The **access token** is used for authenticated API calls.
- The **refresh token** is used to obtain new access tokens without re-login.

3. WebSocket authentication using a **Bearer** token (JWT) Connection Initialization

- JWT-based authentication during the WebSocket handshake. Once authentication succeeds, the mobile app connects to the server via **WebSocket**. The WebSocket layer already identifies the user (typically via headers, token authentication, or parameters).

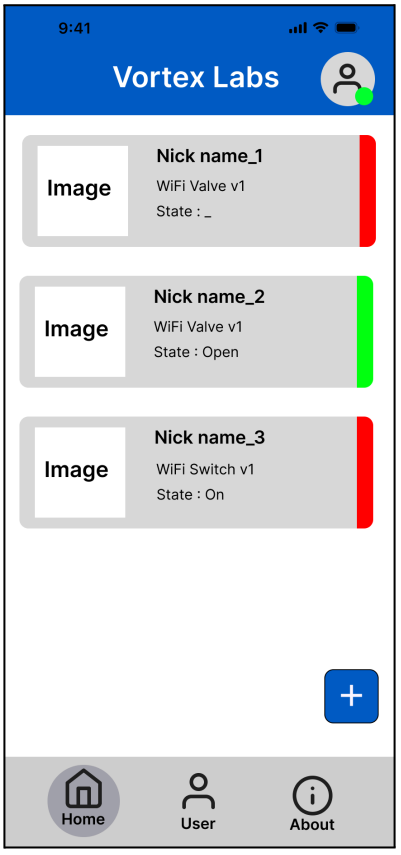
4. Device Data Push (via websocket)

When ui at device list Home page a subscribe msg send to server via websocket.

```
{
 "event": "subscribe",
 "process": "device_list"
}
```

- The server determines which devices belong to the authenticated user.
- It queries the **device info table** based on the user JWT key.
- It sends the initial device data to the mobile app in a structured WebSocket message:

```
{
  "event": "devices_data",
  "user_id": "uid001",
  "devices": 2,
  "device_list": [
     {
        "id": "VA202601001",
        "vwv_name": "MainValve01"
     },
     {
        "id": "VA202601002",
        "vwv_name": "test_2"
     }
  ],
  "timestamp": "2026-02-10T20:27:59+00:00"
}
```



- By extracting device list data Mobile App will display user device list on Home page
- This msg will send to app at every 2 second by websocket. And this process only trigger when user send that **device_list** subscribe msg. After send this msg websocket will continuously send **device_list** msg until the webscoket close or another subscribe msg send (**device_detail**)

User Device Adding process

- In the current system design, **end-users do not have permission** to add or register Vortex devices to their accounts. Device assignment is managed exclusively by the **Vortex Network Admin** through the administrative interface or backend tools

- When a Mobile app receives the initial device data , the App needs to save them on local space to show the user devices even in an offline state. Future help for operate on ESP32 AP mode

WiFi Valve Details Screen - Data visualizing (websocket)

1. User Opens Valve Device

- When the user taps a WiFi valve device layer on the Home screen, the mobile app sends a request to the WebSocket server to fetch the valve's detailed information.

```
{
 "event": "subscribe",
 "process": "device_detail",
 "device_id": "VA202601003"
}
```

2. Server Response Messages

After receiving the request, the server check the user and devices and sends **two types of messages**:

- **Basic Valve Data**

```
{
   "event": "device_basic_detail",
   "device_id": "VA202601003",
   "data": {
      "id": "VA202601003",
      "user_id": "uid002",
      "vwv_name": "valve_1",
      "vwv_version": "v_1.0",
      "vwv_last_seen": null,
      "vwv_is_close": 0,
      "vwv_is_open": 0,
      "vwv_ol_active": 0,
      "vwv_ol_state": 0,
      "vwv_cl_active": 0,
      "vwv_cl_state": 0,
      "vwv_pos": 0,
      "user_set_pos": 0,
      "user_vwv_pos": 0,
      "user_sensor_ctrl": 0,
      "user_schedule_ctrl": 0,
      "sensor_id": null
   },
   "timestamp": "2026-02-10T20:50:12+00:00"
}
```

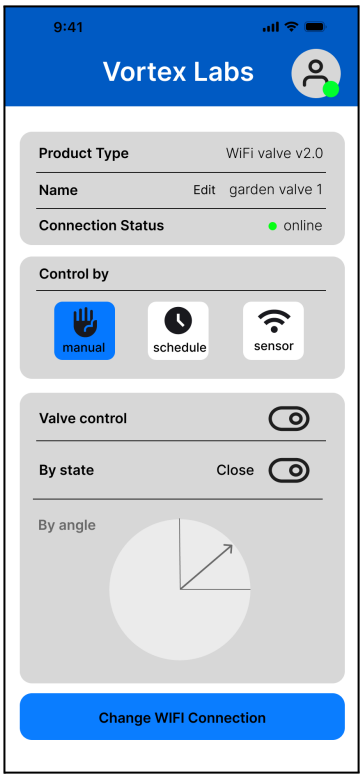
- **Control Valve Data**

```
{

}
```

- When user send **device_detail** subscribe msg to server via websocket it continuously send following msg at every 2 second. This only stops when user stop the websocket connection or send the **device_list** msg to server.
- So when the app start and after the websocket start it continuously send one of the type of data set to the user app.

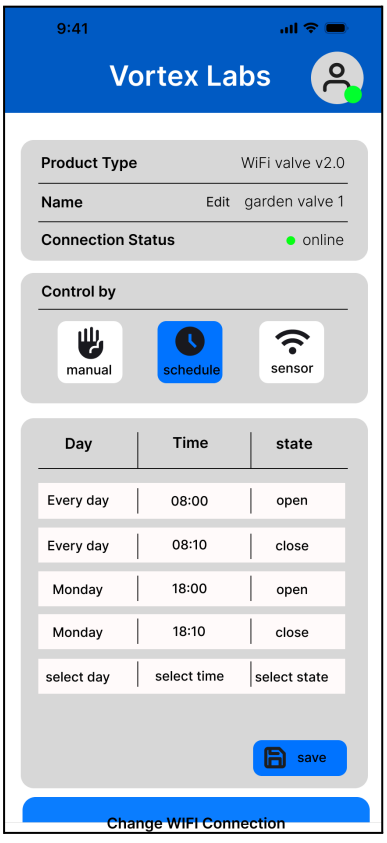
3. UI Rendering Logic in the Mobile App

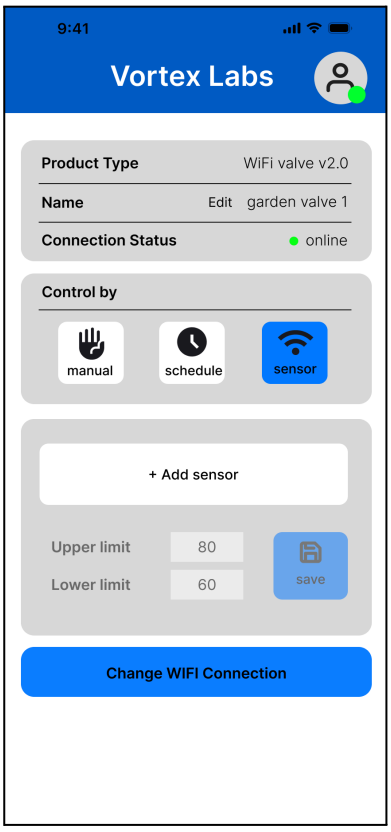


Case 1 — Both `schedule` and `sensor` are FALSE

- **Only basic valve UI is shown**
- App extracts data from **`valve_basic_data` only**
- Sections related to scheduling and sensor settings are **hidden**

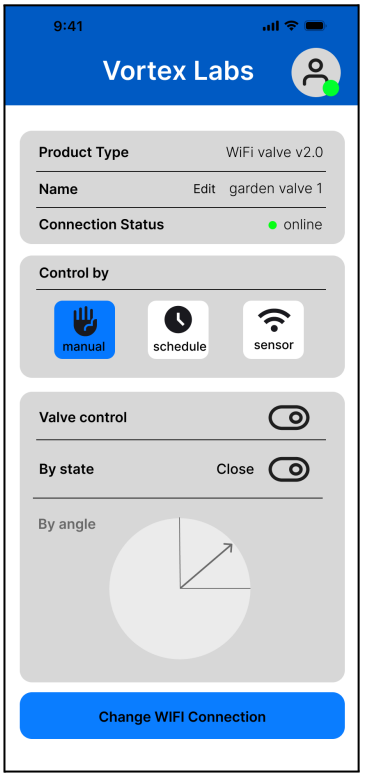
Case 2 — Either `schedule` OR `sensor` is TRUE





WiFi Valve Details - Data editing (REST with JWT token)

1. Valve Basic Data Editing via REST



Valve nickname

- User can edit via an **“Edit”** button in the details section.

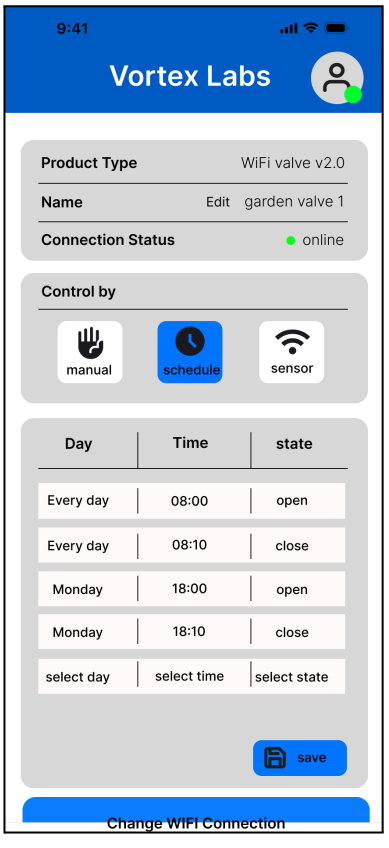
Valve control

- User can operate the valve using:
- Fully Open / Fully Close
- Set a specific angle

When the user edits the valve nickname or changes the angle, This msg send to server **control_device.php** post request

```
{
  "event": "set_valve_basic",
  "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "set_controller": {
        "schedule": false,
        "sensor": false
   },
   "valve_data": {
        "name": "MainValve01",
        "set_angle": true,
        "angle": 45
   },
   "ota_update": false
}
```

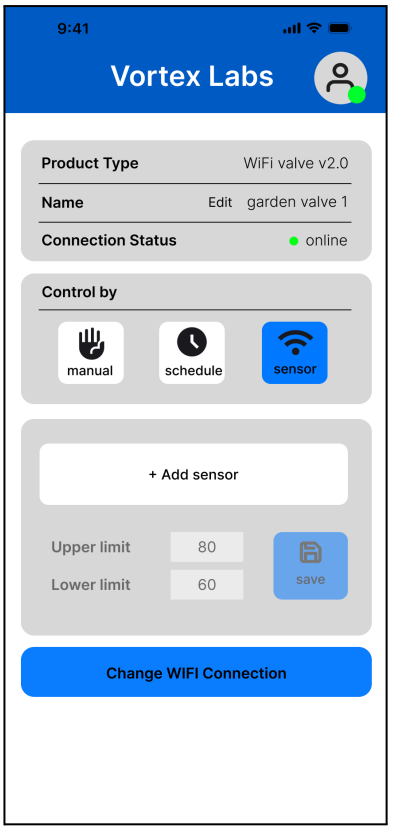
2. Valve Schedule or Sensor Editing via REST



Editable Sections

- Schedule: Add/Edit/Delete schedule entries (day, time, action)
- Sensor: Update sensor IDs and upper/lower limit thresholds

When the user modifies schedule/sensor settings and taps **“Save”**, this message send to server **control_device.php** post request




```
{
   "event": "set_valve_control",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "set_controllerdata": {
        "schedule": true,
        "sensor": false
   },
   "set_sheduledata": {
        "set_shedule": true,
        "schedule_info": [
              {
                    "day": "Monday",
                    "open": "08:00",
                    "close": "08.20"
              },
              {
                    "day": "Monday",
                    "open": "18:00",
                    "close": "18.20"
              }
        ]
   },
   "set_sensordata": {
        "sensor_id": "sensor-01",
        "upper_limit": 80,
        "upper_limit": 30
   }
}
```

If update success this will reply

```
{
   "success": true,
   "message": "Valve control configuration updated"
}
```

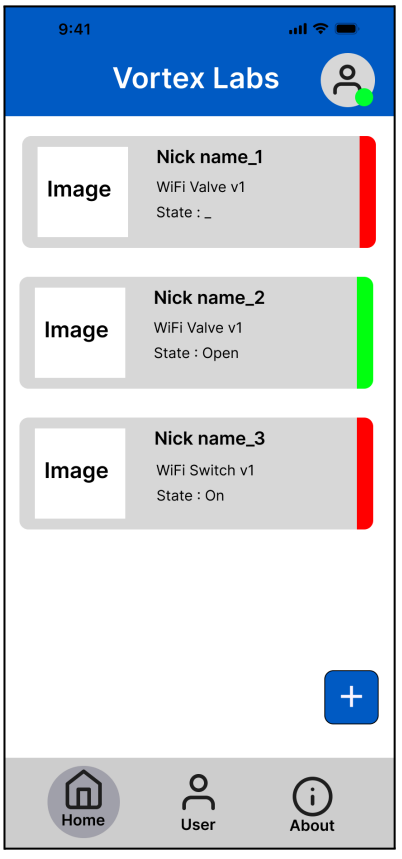
Mobile App And ESP32 Communication (Websocket)

Connection Establishment

1. Identify the Device

- When the ESP32 is in **AP mode**, the user must first connect their mobile device to the ESP32 Wi-Fi network.
- Then, the user opens the **Vortex Lab mobile app**.
  The app checks the connected Wi-Fi SSID and identifies whether it is connected to a **Vortex device Wi-Fi**. If confirmed, the app sends a request to the ESP32 asking for device information.

```
{
        "event": "request_device_info",
        "timestamp": "2025-01-15T10:30:00Z",
        "user_id": "user_id",
        "passkey": "key"
}
```



2. ESP32 response

- The ESP32 reads the request and, if authorized, responds with the device information.

```
{
        "event": "device_info",
        "timestamp": "2025-01-15T10:30:00Z",
        "device_id": "dev0016"
}
```

- On the mobile app side, the app already knows which **device IDs belong to the user**, as this data is stored locally.
- On the **Home screen**, devices are displayed:
    - Devices in **offline mode** are indicated with a **red line**

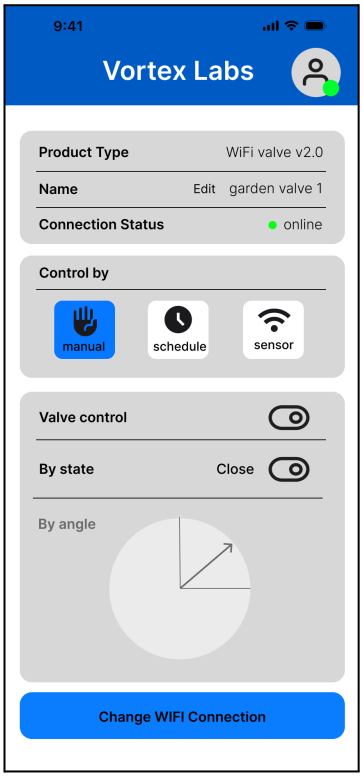
    - If the connected ESP32 device belongs to the user, it is indicated with a **green line**
- After this, the user can control the **Vortex Labs Wi-Fi valve** in **offline mode**.

WiFi Valve Details Screen - Data visualizing

1. User Opens a Valve Device

- When the user taps a Wi-Fi valve device tile on the Home screen, the mobile app sends a message to the ESP32. Need to send this to get valve data

```
{
 "event": "device_basic_info",
 "timestamp": "2025-01-15T10:30:00Z",
 "data": {
        "user_id": "useer001",
        "device_id": "dev0016",
        "device_name": "home valve"
 }
}
```



2. ESP32 Response Messages

- After receiving the request, the ESP32 responds with the valve's current data.

```
{
  "event": "valve_data",
  "timestamp": "2025-01-15T10:30:00Z",
  "device_id": "dev0016",
  "get_controller": {
        "schedule": true,
        "sensor": false
  },
  "get_valvedata": {
        "angle": 45,
        "is_open": true,
        "is_close": false
  },
  "get_limitdata": {
        "is_open_limit": true,
        "open_limit": false,
        "is_close_limit": false,
```

```
        "close_limit": true
    },
    "Error": ""                    }
```

WiFi Valve Details Screen - Data editing

1. Valve Basic Data Editing

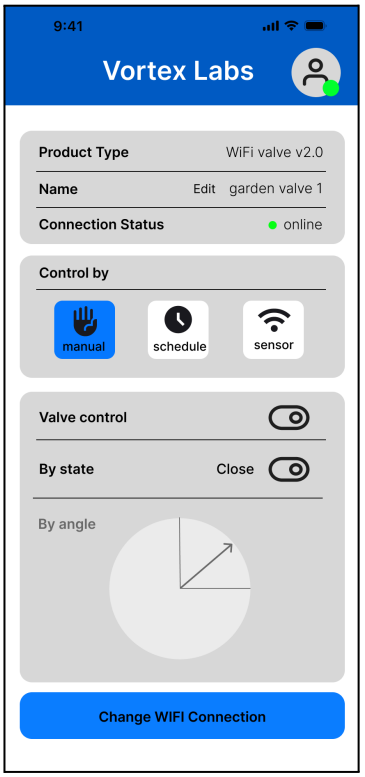
The same **Valve Details screen** is used, but the user is only allowed to control the valve.

Valve control

- Fully Open / Fully Close
- Set a specific angle

When the user edits the **valve nickname** or changes the **valve angle**, the following message is sent to the ESP32.

```
{
  "event": "set_valve_basic",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "set_controller": {
        "schedule": false,
        "sensor": false
   },
   "valve_data": {
        "name": "MainValve01",
        "set_angle": true,
        "angle": 45
   },
   "ota_update": false
}
```



2. Configure the Valve STA mode wifi credentials

By clicking **Change Wi-Fi Connection**, the user can update the valve's **STA mode Wi-Fi credentials**.

A popup appears requesting Wi-Fi details. When the user clicks the **Change** button, the following message is sent to the ESP32.


```
{
  "event": "set_valve_wifi",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "wifi_data": {
        "ssid": "myNetWork",
        "password":"1234"
   }
}
```

WebServer And ESP32 Communication (MQTT)

MQTT topic structure

- The base topic for all Wi-Fi valve communication starts with:

  vortex_device/wifi_valve/{device_id}/

- Main Valve Topics

  vortex_device/wifi_valve/{device_id}/status
  vortex_device/wifi_valve/{device_id}/state_data
  vortex_device/wifi_valve/{device_id}/error
  vortex_device/wifi_valve/{device_id}/cmd_data
  vortex_device/wifi_valve/{device_id}/control_data

Topic Descriptions and Message Flow

1. Device Status Topic

Topic:
vortex_device/wifi_valve/{device_id}/status

Published by: ESP32 device

Subscribed by: Web server

Message:

```
{
   "event": "valve_status",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "status": "online",
}
```

Description:
When the ESP32 is running and connected to the MQTT broker, the device publishes its **online status** to this topic

2. Valve State Data Topic

Topic:
vortex_device/wifi_valve/{device_id}/state_data

Published by: ESP32 device

Subscribed by: Web server

Message:


```
{
   "event": "valve_basic_data",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "get_controller": {
        "schedule": true,
        "sensor": false
   },
   "get_valvedata": {
        "angle": 45,
        "is_open": true,
        "is_close": false
   },
   "get_limitdata": {
        "is_open_limit": true,
        "open_limit": false,
        "is_close_limit": false,
        "close_limit": true
   },
   "Error": ""                    }
```


Description:
The ESP32 publishes **basic valve state data** to this topic.
Updates can be sent:

- Periodically (for example, every second), or
- Only when the valve data changes

3. Device Error Topic

Topic:
vortex_device/wifi_valve/{device_id}/error

Published by: ESP32 device

Subscribed by: Web server

Message:

```
{
   "event": "valve_error",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "error": "",
}
```

Description:
If an error occurs on the ESP32 device, error information is published to this topic.

4. Command Data Topic

Topic:
vortex_device/wifi_valve/{device_id}/cmd_data

Published by: Web server

Subscribed by: ESP32 device

Message:

```
{
  "event": "set_valve_basic",
   "timestamp": "2025-01-15T10:30:00Z",
   "device_id": "dev0016",
   "set_controller": {
        "schedule": false,
        "sensor": false
   },
   "valve_data": {
        "name": "MainValve01",
        "set_angle": true,
        "angle": 45
   },
   "ota_update": false
}
```

Description:
Basic valve command data from the **database or application logic** is

published to this topic. The ESP32 listens to this topic and executes the received commands.

5. Control Data Topic

Topic:

vortex_device/wifi_valve/{device_id}/control_data

Published by: Web server

Subscribed by: ESP32 device

Message:


```
{
  "event": "set_valve_control",
  "timestamp": "2025-01-15T10:30:00Z",
  "device_id": "dev0016",
  "set_controllerdata": {
      "schedule": true,
      "sensor": false
  },
  "set_sheduledata": {
      "set_shedule": true,
      "schedule_info": [
          {
              "day": "Monday",
              "open": "08:00",
              "close": "08.20"
          },
          {
              "day": "Monday",
              "open": "18:00",
              "close": "18.20"
          }
      ]
  },
  "set_sensordata": {
      "sensor_id": "sensor-01",
      "upper_limit": 80,
      "lower_limit": 30
  }
}
```

Description:

Control-related data from the **database or user actions** is published to this topic. This includes valve control operations such as open, close, or setting a specific angle.