ДОДАТОК А

1

Харківський національний університет радіоелектроніки
Кафедра АПОТ

Розподілена система управління вирощуванням рослин

Виконав
студент групи СКСм-20-1
Дегтяр Олександр Миколайович

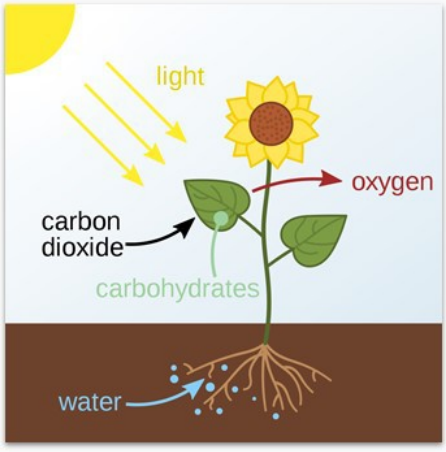
Керівник
проф. каф. АПОТ
Чумаченко С.В.

Харків 2021

2

Умови вирощування

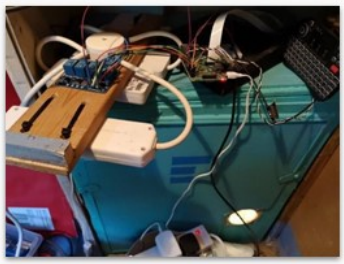
- Поживні речовини
- Вода
- Світло
- Повітря



Актуальність

PiGrow

- Відсутність тестів
- Неактуальна документація
- Налаштування за допомогою безпосереднього підключення



Click&Grow

- Конструктивні недоліки
- Висока вартість

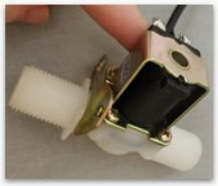
Вимоги до системи

- Автоматична іригація чотирьох окремих зон
  - Періодична іригація
  - Іригація за рівнем вологості ґрунту
  - Визначення підключеності датчиків вологості ґрунту
- Управління фітоосвітленням
  - Ручне управління користувачем
  - Автоматичне управління за часом
- Управління вентиляцією
  - Ручне управління користувачем
  - Автоматичне управління за часом
  - Налаштування потужності вентиляції
- Віддалене налаштування системи за допомогою мобільного застосунку

Апаратне забезпечення

5

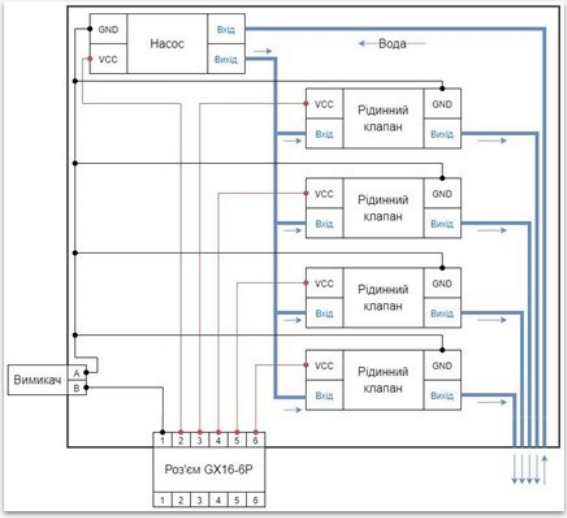
- Raspberry Pi 3B+
- Фітолампа
- Канальний вентилятор
- Перистальтичний насос
- Соленоїдні клапани
- Ємнісні датчики вологості ґрунту
- АЦП ADS1115
- ЄМ-реле
- Блоки живлення 12 В, 34 В
- DC-DC перетворювач 5 В

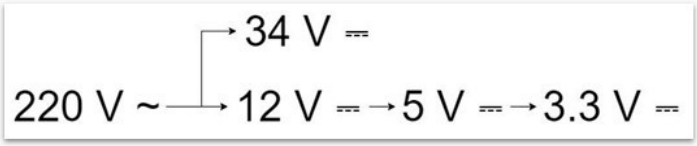

Блок іригації

6

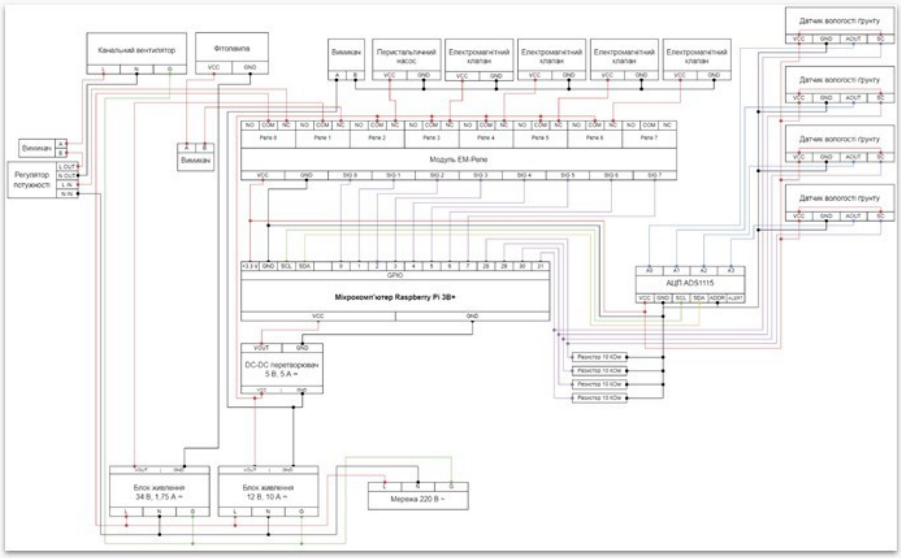
- Ізоляція сантехнічного обладнання
- Демультиплексування потоку воду



Блок управління · Живлення

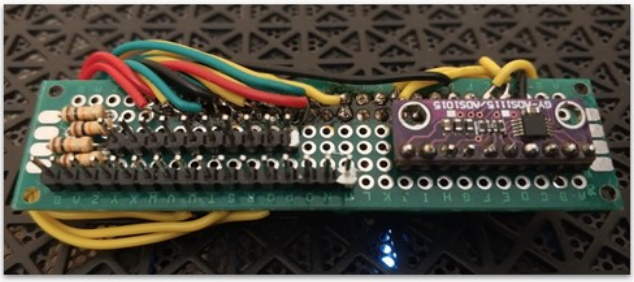


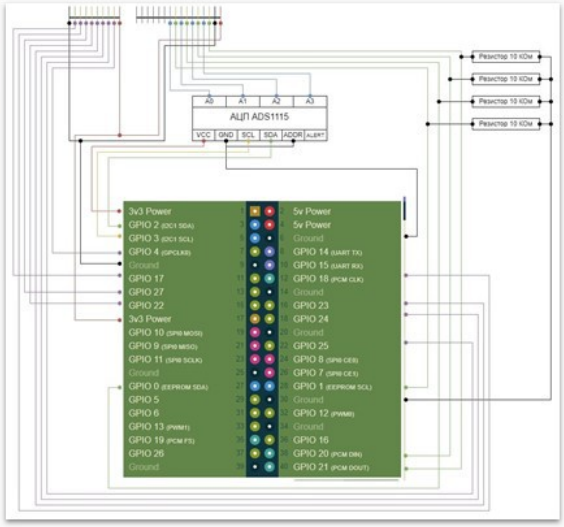
Пристрій · Схема підключення



Пристрій · Плати розширення

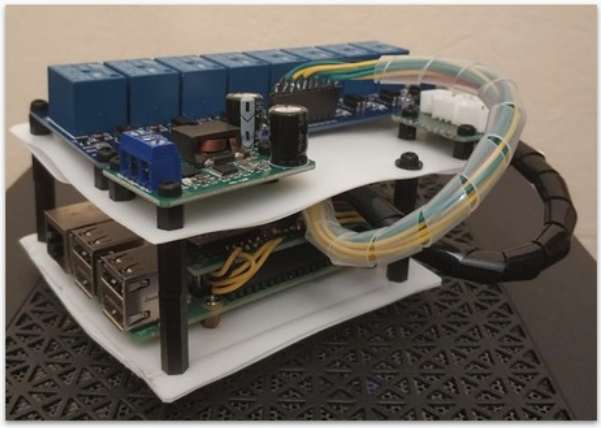
9





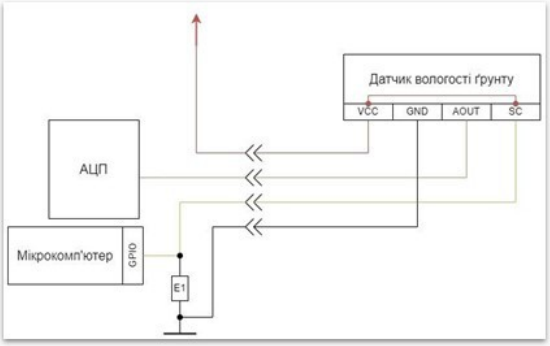
Пристрій · Компонування

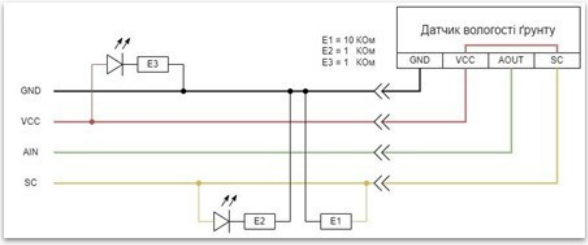
10

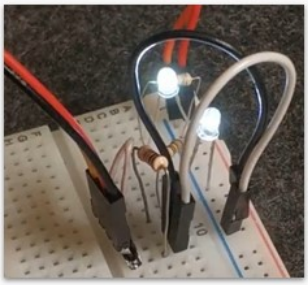


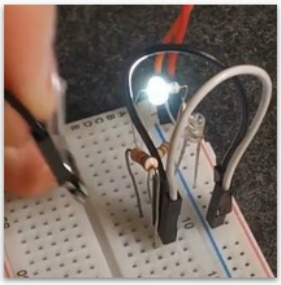
11

Модифікація датчиків · Підключеність



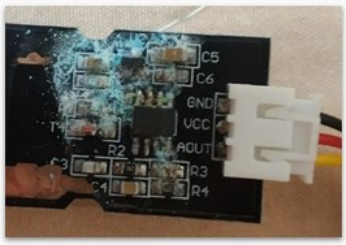


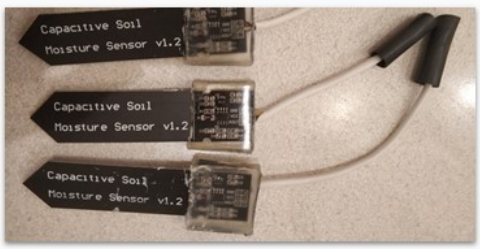




12

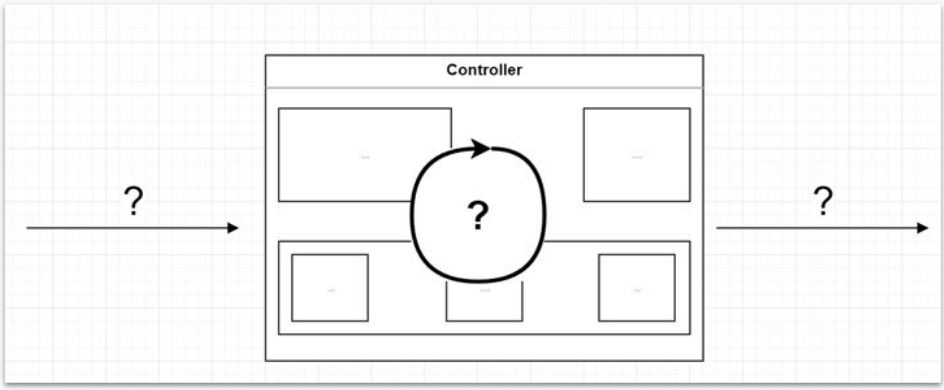
Модифікація датчиків · Ізоляція компонентів





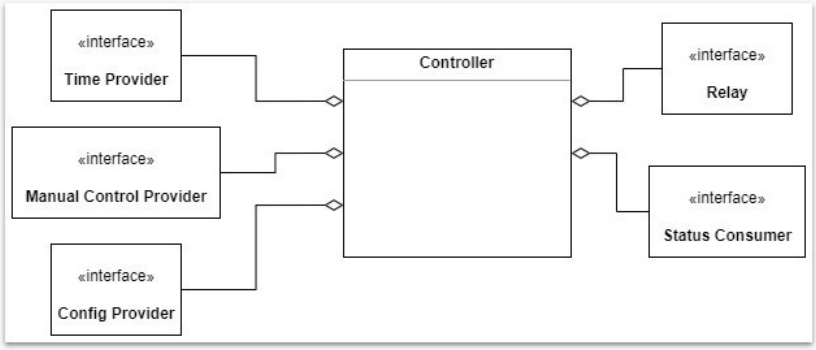
13

Зменшення зв’язності · Проблема



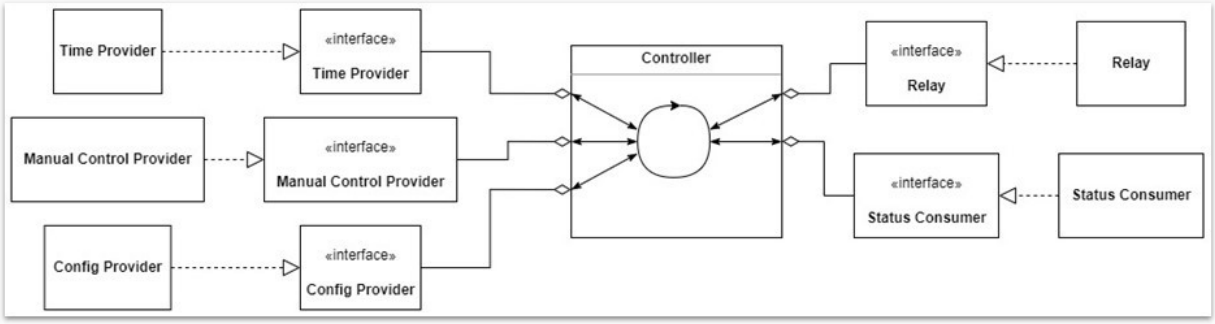
14

Зменшення зв’язності · Інтерфейси



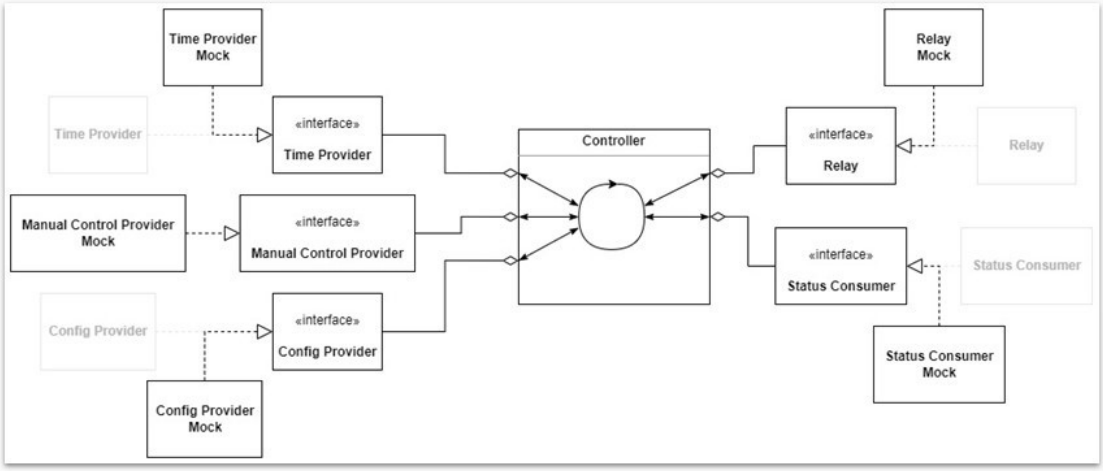
15

Зменшення зв’язності · Залежності

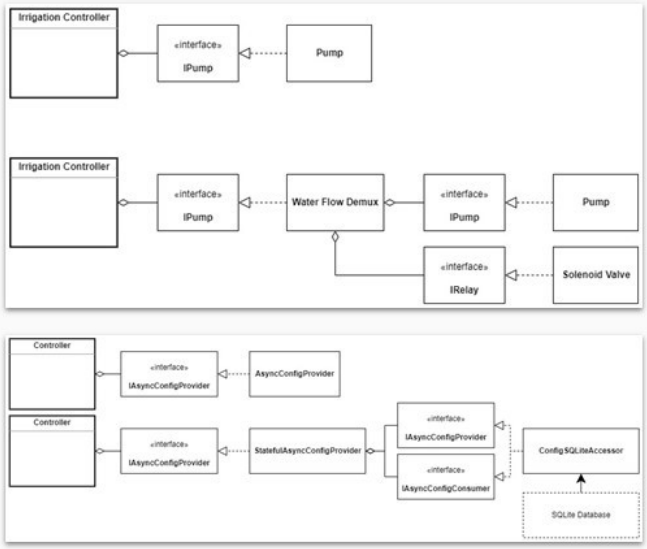


16

Зменшення зв’язності · Автоматизоване тестування



Зменшення зв'язності · Розширення функціональності



Програмні засоби

- Raspberry Pi OS
- C++17
- CMake
- gRPC
- SQLite 3
- GoogleTest

- Android
- Kotlin
- Gradle

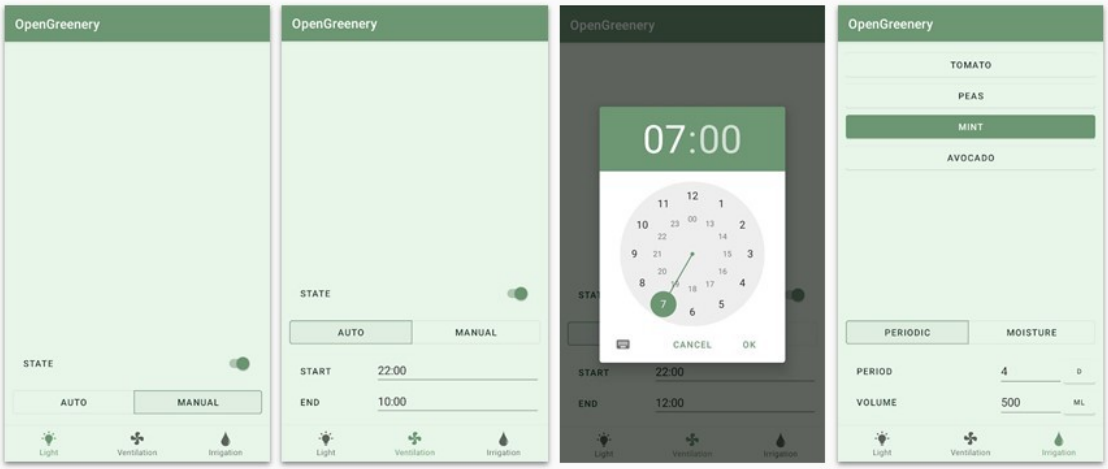
- GitHub Actions

19

Архітектура програмного забезпечення

20

Мобільний застосунок



21

Неперервна інтеграція

Parameterized relay active level
Device Software #36: Pull request #23 opened by Alexander-Degtyar
device/low-level-relay

Android app: Time pickers
Android Application #9: Pull request #26 opened by Alexander-Degtyar
android/time-pickers

README.md

OpenGreenery

Device Software passing Android Application passing

Plants growing automated system

22

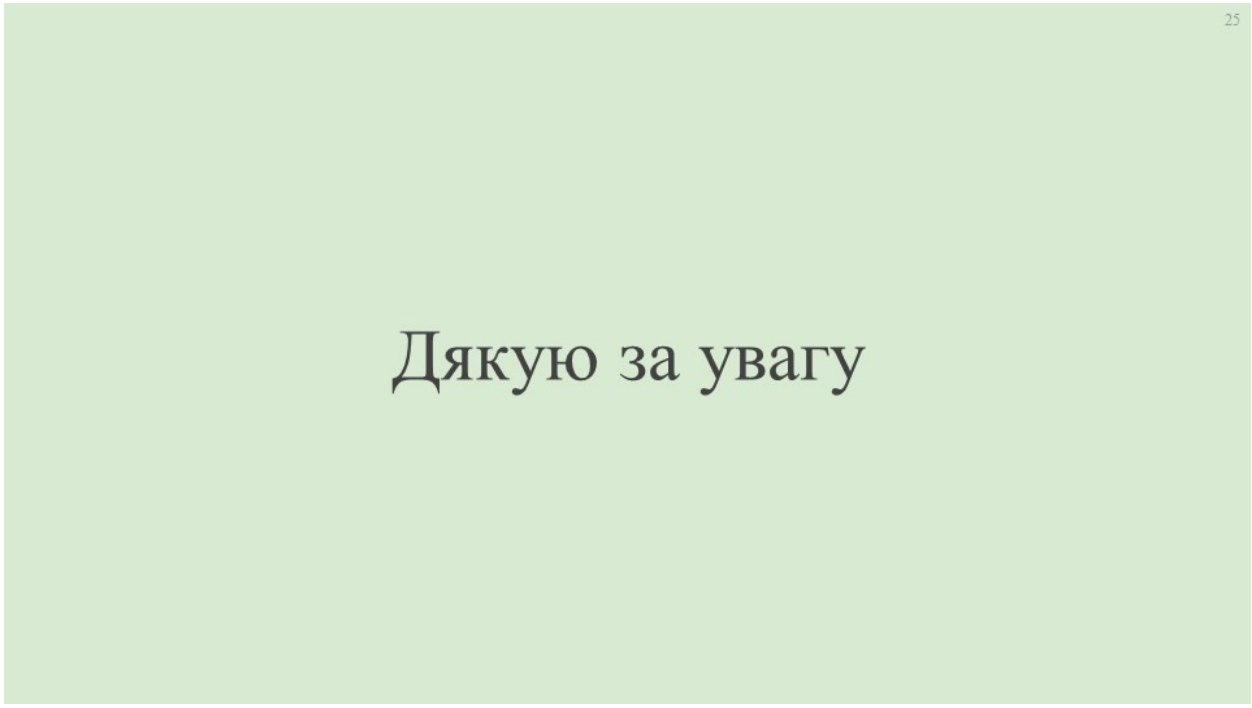
Експериментальне середовище

Розвиток

- Розробка печатної плати розширення
- Веб застосунок
- Виділення аналізу та іригації ґрунту у окремий пристрій
- Виявлення несправностей сантехнічного обладнання
- Користувацькі повідомлення

Висновки

- Розподіленість системи дозволяє знизити вартість апаратного забезпечення пристрою
- Описано метод зменшення зв'язності програмних сутностей
- Реалізовано автоматичні тести
- Налаштовано сервіс неперервної інтеграції
- Збільшено термін служби датчиків вологості ґрунту
- Реалізовано спосіб визначення підключеності датчика до пристрою
- Зменшено вартість сантехнічного обладнання шляхом демультиплексування потоку води
- Реалізовано автоматичну іригацію 4-х зон у періодичному режимі або за вологістю ґрунту
- Реалізовано управління фітоосвітленням та вентиляцією у ручному або автоматичному режимі
- Розроблено мобільний застосунок для налаштування системи
- Обладнано експериментальне середовище


25
Дякую за увагу

ДОДАТОК Б

Б.1 Файл src/device/docker/x86/og-build/Dockerfile

```
FROM alexdegtyar/og-dependencies:x86-v1

ENV BRANCH="develop"
ENV ARTIFACTS=""
ENV ARTIFACTS_FOLDER="/var/og-artifacts"

ENV PROJECT_ROOT="OpenGreenery/src/device"
ENV BUILD_DIRECTORY="${PROJECT_ROOT}/build"

CMD git clone --recursive -b ${BRANCH} --single-branch \
    https://github.com/OpenGreenery/OpenGreenery.git && \
    mkdir -p ${BUILD_DIRECTORY} && \
    cmake -S ${PROJECT_ROOT} -B ${BUILD_DIRECTORY} && \
    cmake --build ${BUILD_DIRECTORY} && \
    for f in ${ARTIFACTS}; do \
        echo Copy artifact ${f}; \
        cp ${BUILD_DIRECTORY}/${f} ${ARTIFACTS_FOLDER}; \
    done
```

Б.2 Файл src/device/docker/x86/og-dependencies/Dockerfile

```
FROM ubuntu:20.04
# avoid timezone selection:
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && \
    apt-get install -y \
    # OpenGreenery
    git \
    cmake \
    build-essential \
    qt5-default \
    libqt5charts5-dev \
    # gRPC
    build-essential autoconf libtool pkg-config

# Install gRPC
WORKDIR /tmp
RUN      git      clone      --recurse-submodules      -b      v1.41.0
https://github.com/grpc/grpc
RUN cd grpc && mkdir -p cmake/build && cd cmake/build && \
    cmake -DgRPC_INSTALL=ON -DgRPC_BUILD_TESTS=OFF ../.. && \
    cmake --build . && make install
```

Б.3 Файл src/device/docker/x86/og-test/Dockerfile

```
FROM alexdegtyar/og-dependencies:x86-v1

ENV ARTIFACTS_FOLDER="/var/og-artifacts"

CMD ${ARTIFACTS_FOLDER}/tests
```

Б.4 Файл src/device/docker/x86/og-test/Dockerfile

```
FROM alexdegtyar/og-dependencies:x86-v1

ENV ARTIFACTS_FOLDER="/var/og-artifacts"

CMD ${ARTIFACTS_FOLDER}/tests
```

Б.4 Файл .github/workflows/device-sw.yml

```
name: Device Software
on:
  push:
    paths:
      - 'src/device/**'
  pull_request:
    paths:
      - 'src/device/**'

  workflow_dispatch:

jobs:
  build:
    runs-on: ubuntu-latest
    env:
      ARTIFACTS_DIR: ${{github.workspace}}/og-artifacts
    steps:
    - name: Determine branch to build
      id: vars
      # Source branch of pull request or to which commit pushed
      run: |
        if [[ ${{ github.event_name }} == 'pull_request' ]]
        then
          BRANCH="${{github.head_ref}}"
        elif [[ ${{ github.event_name }} == 'push' ]]
        then
          BRANCH="${GITHUB_REF#refs/heads/}"
        elif [[ ${{ github.event_name }} == 'workflow_dispatch' ]]
        then
          BRANCH="${GITHUB_REF#refs/heads/}"
        else
          BRANCH="${{ github.event.repository.default_branch }}"
        fi
        echo ::set-output name=branch::${BRANCH}
    - name: Pull Docker images
      run: |
        docker pull alexdegtyar/og-build:x86-latest
        docker pull alexdegtyar/og-test:x86-latest
    - name: Create artifacts directory
      run: mkdir ${{ env.ARTIFACTS_DIR }}
    - name: Build project
      run: |
        echo Build branch ${{ steps.vars.outputs.branch }}
        docker run \
        -e BRANCH=${{ steps.vars.outputs.branch }} \
        -e ARTIFACTS="tests/tests" \
        -v ${{ env.ARTIFACTS_DIR }}:/var/og-artifacts \
        alexdegtyar/og-build:x86-latest
    - name: Run tests
      run: |
        docker run \
        -v ${{ env.ARTIFACTS_DIR }}:/var/og-artifacts \
```

```
        alexdegtyar/og-test:x86-latest
```

Б.5 Файл .github/workflows/android.yml

```
name: Android Application

on:
  push:
    paths:
      - 'src/android/**'
  pull_request:
    paths:
      - 'src/android/**'
  workflow_dispatch:

defaults:
  run:
    working-directory: src/android

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v2
      with:
        java-version: '11'
        distribution: 'adopt'
        cache: gradle

    - name: Grant execute permission for gradlew
      run: chmod +x gradlew
    - name: Build with Gradle
      run: ./gradlew build
```

Б.6 Файл src/device/lib/dataflow/inc/common/IAsyncProvider.hpp

```
#pragma once

#include <functional>

namespace open_greenery::dataflow::common
{

template<typename T>
using AsyncReceive = std::function<void(T)>;

template<typename T>
class IAsyncProvider
{
public:
    virtual ~IAsyncProvider() = default;

    virtual void onUpdate(AsyncReceive<T> receive) = 0;
};
```

```
}
```

Б.7 Файл src/device/lib/dataflow/inc/common/IAsyncReceiver.hpp

```
#pragma once

#include <functional>

namespace open_greenery::dataflow::common
{

template <typename T>
using AsyncProvide = std::function<T()>;

template <typename T>
class IAsyncReceiver
{
public:
    virtual ~IAsyncReceiver() = default;

    virtual void onRequest(AsyncProvide<T> provide) = 0;
};

}
```

Б.8 src/device/lib/dataflow/inc/common/IOptionalProvider.hpp

```
#ifndef I_OPTIONAL_PROVIDER_HPP
#define I_OPTIONAL_PROVIDER_HPP

#include <optional>

namespace open_greenery::dataflow::common
{

template<class T>
class IOptionalProvider
{
public:
    virtual std::optional<T> get() = 0;

    IOptionalProvider() = default;
    virtual ~IOptionalProvider() = default;
};

}

#endif //I_OPTIONAL_PROVIDER_HPP
```

Б.9 Файл src/device/lib/dataflow/inc/common/IProvider.hpp

```
#ifndef I_PROVIDER_HPP
#define I_PROVIDER_HPP

namespace open_greenery::dataflow::common
{

template<class T>
class IProvider
{
```

```
public:
    virtual T get() = 0;

    IProvider() = default;
    virtual ~IProvider() = default;
};

}

#endif //I_PROVIDER_HPP
```

Б.10 Файл src/device/lib/dataflow/inc/common/IReceiver.hpp

```
#ifndef I_RECEIVER_HPP
#define I_RECEIVER_HPP

namespace open_greenery::dataflow::common
{

template<class T>
class IReceiver
{
public:
    virtual void set(T value) = 0;

    IReceiver() = default;
    virtual ~IReceiver() = default;
};

}

#endif //I_RECEIVER_HPP
```

Б.11 Файл src/device/lib/dataflow/inc/irrigation/IrrigationConfigRecord.hpp

```
#ifndef IRRIGATION_CONFIG_RECORD_HPP
#define IRRIGATION_CONFIG_RECORD_HPP

#include <chrono>
#include <string>
#include <open_greenery/gpio/PinId.hpp>

namespace open_greenery::dataflow::irrigation
{

struct IrrigationConfigRecord
{
    open_greenery::gpio::PinId pin;
    std::int16_t dry;
    std::int16_t wet;
    std::uint16_t watering_volume;
    std::chrono::seconds watering_period;
    std::string soil_moisture_sensor;
};

}

#endif //IRRIGATION_CONFIG_RECORD_HPP
```

Б.12 Файл src/device/lib/dataflow/inc/irrigation/SensorRecord.hpp

```
#ifndef SENSOR_RECORD_HPP
#define SENSOR_RECORD_HPP

#include <cstdint>
#include <QDateTime>

namespace open_greenery::dataflow::irrigation
{

struct SensorRecord
{
    QDateTime timestamp;
    std::int16_t value;

    bool operator == (const SensorRecord & r) const
    {
        return (timestamp == r.timestamp) && (value == r.value);
    }

    bool operator != (const SensorRecord & r) const
    {
        return !(*this==r);
    }
};

}

#endif //SENSOR_RECORD_HPP
```

Б.13 Файл src/device/lib/dataflow/inc/irrigation/Participants.hpp

```
#pragma once

#include <cstdint>
#include <vector>
#include "SensorRecord.hpp"
#include "IrrigationConfigRecord.hpp"
#include <open_greenery/dataflow/common/IOptionalProvider.hpp>
#include <open_greenery/dataflow/common/IProvider.hpp>
#include <open_greenery/dataflow/common/IReceiver.hpp>

namespace open_greenery::dataflow::irrigation
{

class IIrrigationConfigDataProvider
{
public:
    virtual std::vector<IrrigationConfigRecord> read() = 0;
    virtual IrrigationConfigRecord read(open_greenery::gpio::PinId _pin)
= 0;

    IIrrigationConfigDataProvider() = default;
    virtual ~IIrrigationConfigDataProvider() = default;
};
using                    IIrrigationDataReceiver                    =
open_greenery::dataflow::common::IReceiver<IrrigationConfigRecord>;

using                  IOptionalSensorReadProvider                  =
open_greenery::dataflow::common::IOptionalProvider<std::int16_t>;
using                      ISensorReadProvider                      =
open_greenery::dataflow::common::IProvider<std::int16_t>;
```

```
class ISensorDataProvider
{
public:
     virtual std::vector<SensorRecord> read(QDateTime _from, QDateTime
_to) const = 0;

    ISensorDataProvider() = default;
    virtual ~ISensorDataProvider() = default;
};

class ISensorDataReceiver
{
public:
    virtual void write(std::int16_t _data) = 0;
    virtual void write(SensorRecord _record) = 0;
    virtual void write(QDateTime _timestamp, std::int16_t _data) = 0;

    ISensorDataReceiver() = default;
    virtual ~ISensorDataReceiver() = default;
};

}
```

Б.14 Файл src/device/lib/dataflow/inc/relay/ConfigRecord.hpp

```
#ifndef LIGHT_CONFIG_RECORD_HPP
#define LIGHT_CONFIG_RECORD_HPP

#include <QTime>

namespace open_greenery::dataflow::relay
{

struct Config
{
    QTime day_start;
    QTime day_end;
};

}

#endif //LIGHT_CONFIG_RECORD_HPP
```

Б.15 Файл src/device/lib/dataflow/inc/relay/Control.hpp

```
#ifndef LIGHT_CONTROL_HPP
#define LIGHT_CONTROL_HPP

namespace open_greenery::dataflow::relay
{

enum class Control : std::uint8_t
{
    ENABLE, DISABLE, TOGGLE
};

}

#endif //LIGHT_CONTROL_HPP
```

Б.16 Файл src/device/lib/dataflow/inc/relay/Mode.hpp

```
#ifndef LIGHT_MODE_HPP
#define LIGHT_MODE_HPP

namespace open_greenery::dataflow::relay
{

enum class Mode : std::uint8_t
{
    AUTO, MANUAL
};

}

#endif //LIGHT_MODE_HPP
```

Б.17 Файл src/device/lib/dataflow/inc/relay/Participants.hpp

```
#pragma once

#include "ConfigRecord.hpp"
#include "Control.hpp"
#include "Mode.hpp"
#include "ServiceStatus.hpp"
#include <open_greenery/dataflow/common/IAsyncProvider.hpp>
#include <open_greenery/dataflow/common/IAsyncReceiver.hpp>
#include <open_greenery/dataflow/common/IOptionalProvider.hpp>
#include <open_greenery/dataflow/common/IProvider.hpp>
#include <open_greenery/dataflow/common/IReceiver.hpp>

namespace open_greenery::dataflow::relay
{

using                          IConfigProvider                          =
open_greenery::dataflow::common::IProvider<Config>;
using                       IConfigOptionalProvider                     =
open_greenery::dataflow::common::IOptionalProvider<Config>;
using                          IConfigReceiver                          =
open_greenery::dataflow::common::IReceiver<Config>;
using                        IAsyncConfigProvider                       =
open_greenery::dataflow::common::IAsyncProvider<Config>;
using                        IAsyncConfigReceiver                       =
open_greenery::dataflow::common::IAsyncReceiver<Config>;

using                       IManualControlProvider                      =
open_greenery::dataflow::common::IProvider<Control>;
using                  IManualControlOptionalProvider                   =
open_greenery::dataflow::common::IOptionalProvider<Control>;
using                       IManualControlReceiver                      =
open_greenery::dataflow::common::IReceiver<Control>;
using                   IAsyncManualControlProvider                     =
open_greenery::dataflow::common::IAsyncProvider<Control>;
using                   IAsyncManualControlReceiver                     =
open_greenery::dataflow::common::IAsyncReceiver<Control>;

using IModeProvider = open_greenery::dataflow::common::IProvider<Mode>;
using                    IModeProviderOptionalProvider                  =
open_greenery::dataflow::common::IOptionalProvider<Mode>;
using IModeReceiver = open_greenery::dataflow::common::IReceiver<Mode>;
using                         IAsyncModeProvider                        =
open_greenery::dataflow::common::IAsyncProvider<Mode>;
```

```
    using                          IAsyncModeReceiver                          =
open_greenery::dataflow::common::IAsyncReceiver<Mode>;

    using                         IRelayStatusProvider                         =
open_greenery::dataflow::common::IProvider<bool>;
    using                  IRelayStatusOptionalProvider                        =
open_greenery::dataflow::common::IOptionalProvider<bool>;
    using                          IRelayStatusReceiver                        =
open_greenery::dataflow::common::IReceiver<bool>;
    using                     IAsyncRelayStatusProvider                        =
open_greenery::dataflow::common::IAsyncProvider<bool>;
    using                     IAsyncRelayStatusReceiver                        =
open_greenery::dataflow::common::IAsyncReceiver<bool>;

    using                        IServiceStatusProvider                        =
open_greenery::dataflow::common::IProvider<ServiceStatus>;
    using                 IServiceStatusOptionalProvider                       =
open_greenery::dataflow::common::IOptionalProvider<ServiceStatus>;
    using                        IServiceStatusReceiver                        =
open_greenery::dataflow::common::IReceiver<ServiceStatus>;
    using                   IAsyncServiceStatusProvider                        =
open_greenery::dataflow::common::IAsyncProvider<ServiceStatus>;
    using                   IAsyncServiceStatusReceiver                        =
open_greenery::dataflow::common::IAsyncReceiver<ServiceStatus>;

    }
```

Б.18 Файл src/device/lib/dataflow/inc/relay/ServiceStatus.hpp

```
#pragma once

#include <optional>
#include "Mode.hpp"
#include "ConfigRecord.hpp"

namespace open_greenery::dataflow::relay
{

struct ServiceStatus
{
    Mode mode {};
    bool relay_enabled {};
    Config config;
};

}
```

Б.19 Файл src/device/lib/dataflow/inc/relay/ServiceStatus.hpp

```
#pragma once

#include <optional>
#include "Mode.hpp"
#include "ConfigRecord.hpp"

namespace open_greenery::dataflow::relay
{

struct ServiceStatus
{
    Mode mode {};
```

```
        bool relay_enabled {};
        Config config;
    };

    }
```

Б.20 Файл src/device/lib/dataflow/inc/time/Participants.hpp

```
    #pragma once

    #include <QTime>
    #include <QDateTime>
    #include <open_greenery/dataflow/common/IProvider.hpp>

    namespace open_greenery::dataflow::time
    {

    using                        ICurrentTimeProvider                        =
open_greenery::dataflow::common::IProvider<QTime>;
    using                     ICurrentDateTimeProvider                       =
open_greenery::dataflow::common::IProvider<QDateTime>;

    }
```

Б.21 Файл src/device/lib/dataflow/CMakeLists.txt

```
    project( dataflow )

    add_library( ${PROJECT_NAME}    INTERFACE )
    target_include_directories( ${PROJECT_NAME}    INTERFACE inc )
    find_package( Qt5 REQUIRED COMPONENTS    Core )
    target_link_libraries( ${PROJECT_NAME}    INTERFACE Qt5::Core gpio)
```

Б.22 Файл src/device/lib/rpc/relay/inc/open_greenery/rpc/relay/Client.hpp

```
    #ifndef LIGHT_PROXY_CLIENT_HPP
    #define LIGHT_PROXY_CLIENT_HPP

    #include <memory>
    #include <open_greenery/dataflow/relay/Participants.hpp>
    #include "relay.grpc.pb.h"

    namespace open_greenery::rpc::relay
    {

    class Client :
            public open_greenery::dataflow::relay::IConfigReceiver,
            public open_greenery::dataflow::relay::IManualControlReceiver,
            public open_greenery::dataflow::relay::IModeReceiver
    {
    public:
        explicit Client(const std::string & host);

        // IConfigReceiver
        void set(open_greenery::dataflow::relay::Config record) override;

        // IManualControlReceiver
        void set(open_greenery::dataflow::relay::Control control) override;

        // IModeReceiver
```

```
        void set(open_greenery::dataflow::relay::Mode mode) override;

                        class    RelayStatusOptionalProvider    :    public
open_greenery::dataflow::relay::IRelayStatusOptionalProvider
        {
        public:
            RelayStatusOptionalProvider(std::shared_ptr<Relay::Stub> stub);

            std::optional<bool> get() override;

        private:
            std::shared_ptr<Relay::Stub> m_stub;
        };

                      class    ServiceStatusOptionalProvider    :    public
open_greenery::dataflow::relay::IServiceStatusOptionalProvider
        {
        public:
                  ServiceStatusOptionalProvider(std::shared_ptr<Relay::Stub>
stub);

                std::optional<open_greenery::dataflow::relay::ServiceStatus>
get() override;

        private:
            std::shared_ptr<Relay::Stub> m_stub;
        };

                                  std::unique_ptr<RelayStatusOptionalProvider>
getRelayStatusOptionalProvider() const;

                                std::unique_ptr<ServiceStatusOptionalProvider>
getServiceStatusOptionalProvider() const;
    private:
        std::shared_ptr<grpc::Channel> m_channel;
        std::shared_ptr<Relay::Stub> m_stub;
    };

    }

    #endif //LIGHT_PROXY_CLIENT_HPP
```

Б.23 Файл src/device/lib/rpc/relay/inc/open_greenery/rpc/relay/Server.hpp

```
    #ifndef SERVER_HPP
    #define SERVER_HPP

    #include <optional>
    #include <grpcpp/server.h>
    #include <open_greenery/dataflow/relay/Participants.hpp>
    #include "relay.grpc.pb.h"
    #include "relay.pb.h"

    namespace open_greenery::rpc::relay
    {

    class Server :
            public open_greenery::dataflow::relay::IAsyncConfigProvider,
                                                                    public
open_greenery::dataflow::relay::IAsyncManualControlProvider,
            public open_greenery::dataflow::relay::IAsyncModeProvider,
                                                                    public
```

```
open_greenery::dataflow::relay::IAsyncRelayStatusReceiver,
                                                                    public
open_greenery::dataflow::relay::IAsyncServiceStatusReceiver
      {
      public:
          explicit Server(const std::string & host);

          ~Server() override;

          void wait();

          void shutdown();

          void onConfigUpdate(
                  open_greenery::dataflow::common::AsyncReceive
                          <open_greenery::dataflow::relay::Config> handler);

          void onUpdate(
                  open_greenery::dataflow::common::AsyncReceive
                            <open_greenery::dataflow::relay::Config> receive)
override;

          void onManualControl(
                  open_greenery::dataflow::common::AsyncReceive
                          <open_greenery::dataflow::relay::Control> handler);

          void onUpdate(
                  open_greenery::dataflow::common::AsyncReceive
                            <open_greenery::dataflow::relay::Control> receive)
override;

          void onModeUpdate(
                  open_greenery::dataflow::common::AsyncReceive
                          <open_greenery::dataflow::relay::Mode> handler);

          void onUpdate(
                  open_greenery::dataflow::common::AsyncReceive
                              <open_greenery::dataflow::relay::Mode> receive)
override;

                                                                      void
onStatusRequest(open_greenery::dataflow::common::AsyncProvide<bool> handler);

            void onRequest(open_greenery::dataflow::common::AsyncProvide<bool>
provide) override;

          void onRequest(
                  open_greenery::dataflow::common::AsyncProvide
                              <open_greenery::dataflow::relay::ServiceStatus>
provide) override;

      private:
          class Service : public Relay::Service
          {
          public:
              // Relay::Service gRPC
              grpc::Status SetConfig(grpc::ServerContext * context,
                                     const open_greenery::rpc::relay::Config *
request,
                                           google::protobuf::Empty * response)
override;

              grpc::Status ManualControl(grpc::ServerContext * context,
                                                                      const
```

```
open_greenery::rpc::relay::ManualControlRequest * request,
                                          google::protobuf::Empty * response)
override;

              grpc::Status SetMode(grpc::ServerContext * context,
                                                                        const
open_greenery::rpc::relay::ModeSetting * request,
                                           google::protobuf::Empty * response)
override;

              grpc::Status GetRelayStatus(grpc::ServerContext * context,
                                    const google::protobuf::Empty * request,
                                     open_greenery::rpc::relay::RelayStatus *
response) override;

              grpc::Status GetServiceStatus(grpc::ServerContext * context,
                                              const google::protobuf::Empty *
request,
                                               open_greenery::rpc::relay::Service
Status * response) override;

              open_greenery::dataflow::common::AsyncReceive
                                     <open_greenery::dataflow::relay::Config>
m_config_update_handler;
              open_greenery::dataflow::common::AsyncReceive
                                    <open_greenery::dataflow::relay::Control>
m_manual_control_handler;
              open_greenery::dataflow::common::AsyncReceive
                                       <open_greenery::dataflow::relay::Mode>
m_mode_update_handler;
                             open_greenery::dataflow::common::AsyncProvide<bool>
m_relay_status_request_handler;
              open_greenery::dataflow::common::AsyncProvide
                                 <open_greenery::dataflow::relay::ServiceStatus>
m_service_status_request_handler;
          };

          Service m_service;
          std::unique_ptr<grpc::Server> m_server;
      };

      }

      #endif //SERVER_HPP
```

Б.24 Файл src/device/lib/rpc/relay/protos/relay.proto

```
syntax = "proto3";
import "google/protobuf/empty.proto";

package open_greenery.rpc.relay;
option java_package = "com.open_greenery.mobile";
option java_outer_classname = "RelayProto";

service Relay {
  rpc SetConfig(Config) returns (google.protobuf.Empty) {}
              rpc       ManualControl(ManualControlRequest)       returns
(google.protobuf.Empty) {}
  rpc SetMode(ModeSetting) returns (google.protobuf.Empty) {}
  rpc GetRelayStatus(google.protobuf.Empty) returns (RelayStatus) {}
  rpc GetServiceStatus(google.protobuf.Empty) returns (ServiceStatus) {}
}
```

```
message Config {
  // msecs since start of day
  int32 day_start = 1;
  int32 day_end = 2;
}

message ManualControlRequest {
  enum Control {
    CONTROL_ENABLE = 0;
    CONTROL_DISABLE = 1;
    CONTROL_TOGGLE = 2;
  }
  Control control = 1;
}

message ModeSetting {
  enum Mode {
    MODE_MANUAL = 0;
    MODE_AUTO = 1;
  }
  Mode mode = 1;
}

message RelayStatus {
  bool is_enabled = 1;
}

message ServiceStatus {
  ModeSetting mode_settings = 1;
  RelayStatus relay_status = 2;
  Config config = 3;
}
```

Б.25 Файл src/device/lib/rpc/relay/src/Client.cpp

```
#include "open_greenery/rpc/relay/Client.hpp"

#include <utility>
#include <grpcpp/create_channel.h>
#include <grpcpp/channel.h>
#include <spdlog/spdlog.h>

namespace open_greenery::rpc::relay
{

Client::Client(const std::string & host)
                                        :m_channel(grpc::CreateChannel(host,
grpc::InsecureChannelCredentials())),
         m_stub(Relay::NewStub(m_channel)) {}

void Client::set(open_greenery::dataflow::relay::Config record)
{
    grpc::ClientContext context;
    Config request;
    google::protobuf::Empty response;

    const auto day_start = record.day_start.msecsSinceStartOfDay();
    request.set_day_start(day_start);
    const auto day_end = record.day_end.msecsSinceStartOfDay();
    request.set_day_end(day_end);
```

```
                    auto    TimeStr    =    [](const    QTime    &    t){return
t.toString("hh:mm:ss").toStdString();};
          spdlog::debug("Send relay config: start={}, end={}",
                        TimeStr(record.day_start),
                        TimeStr(record.day_end));

          auto status = m_stub->SetConfig(&context, request, &response);
          if (!status.ok())
          {
              spdlog::warn("Unsuccessful Relay::SetConfig request: {} {}",
                           status.error_code(),
                           status.error_message());
          }
      }

      void Client::set(open_greenery::dataflow::relay::Control control)
      {
          grpc::ClientContext context;
          ManualControlRequest request;
          google::protobuf::Empty response;

          switch (control)
          {
              case open_greenery::dataflow::relay::Control::ENABLE:
                  spdlog::debug("Send manual enable");
                  request.set_control(ManualControlRequest::CONTROL_ENABLE);
                  break;
              case open_greenery::dataflow::relay::Control::DISABLE:
                  spdlog::debug("Send manual disable");
                  request.set_control(ManualControlRequest::CONTROL_DISABLE);
                  break;
              case open_greenery::dataflow::relay::Control::TOGGLE:
                  spdlog::debug("Send manual toggle");
                  request.set_control(ManualControlRequest::CONTROL_TOGGLE);
                  break;
              default:
                  assert(false && "Unknown light::Control type");
          }

          auto status = m_stub->ManualControl(&context, request, &response);
          if (!status.ok())
          {
              spdlog::warn("Unsuccessful Relay::ManualControl request: {} {}",
                           status.error_code(),
                           status.error_message());
          }
      }

      void Client::set(open_greenery::dataflow::relay::Mode mode)
      {
          grpc::ClientContext context;
          ModeSetting request;
          google::protobuf::Empty response;

          switch (mode)
          {
              case open_greenery::dataflow::relay::Mode::MANUAL:
                  spdlog::debug("Send mode setting: MANUAL");
                  request.set_mode(ModeSetting::MODE_MANUAL);
                  break;
              case open_greenery::dataflow::relay::Mode::AUTO:
                  spdlog::debug("Send mode setting: AUTO");
                  request.set_mode(ModeSetting::MODE_AUTO);
                  break;
```

```
            default:
                assert(false && "Unknown relay::Mode type");
        }

        auto status = m_stub->SetMode(&context, request, &response);
        if (!status.ok())
        {
            spdlog::warn("Unsuccessful Relay::SetMode request: {} {}",
                        status.error_code(),
                        status.error_message());
        }
    }

    std::unique_ptr<Client::RelayStatusOptionalProvider>
Client::getRelayStatusOptionalProvider() const
    {
        return std::make_unique<RelayStatusOptionalProvider>(m_stub);
    }

    std::unique_ptr<Client::ServiceStatusOptionalProvider>
Client::getServiceStatusOptionalProvider() const
    {
        return std::make_unique<ServiceStatusOptionalProvider>(m_stub);
    }

    Client::RelayStatusOptionalProvider::RelayStatusOptionalProvider(std::sh
ared_ptr<Relay::Stub> stub)
            :m_stub(std::move(stub)){}

    std::optional<bool> Client::RelayStatusOptionalProvider::get()
    {
        grpc::ClientContext context;
        google::protobuf::Empty request;
        RelayStatus response;

        auto status = m_stub->GetRelayStatus(&context, request, &response);
        if (!status.ok())
        {
               spdlog::warn("Unsuccessful Relay::GetRelayStatus request: {}
{}",
                        status.error_code(),
                        status.error_message());
            return {};
        }

        return response.is_enabled();
    }

    Client::ServiceStatusOptionalProvider::ServiceStatusOptionalProvider(std
::shared_ptr<Relay::Stub> stub)
            :m_stub(std::move(stub)){}

    std::optional<open_greenery::dataflow::relay::ServiceStatus>
Client::ServiceStatusOptionalProvider::get()
    {
        grpc::ClientContext context;
        google::protobuf::Empty request;
        ServiceStatus response;

             auto status = m_stub->GetServiceStatus(&context, request,
&response);
        if (!status.ok())
        {
             spdlog::warn("Unsuccessful Relay::GetServiceStatus request: {}
```

```
{}",
                            status.error_code(),
                            status.error_message());
                return {};
            }

            const open_greenery::dataflow::relay::Mode mode =
                                              (response.mode_settings().mode()   ==
ModeSetting::MODE_MANUAL) ?
                                  open_greenery::dataflow::relay::Mode::MANUAL  :
open_greenery::dataflow::relay::Mode::AUTO;
            const bool relay_enabled = response.relay_status().is_enabled();
            const open_greenery::dataflow::relay::Config config =
                    {QTime::fromMSecsSinceStartOfDay(response.config().day_start
()),
                     QTime::fromMSecsSinceStartOfDay(response.config().day_end()
)};

            open_greenery::dataflow::relay::ServiceStatus service_status;
            service_status.mode = mode;
            service_status.relay_enabled = relay_enabled;
            service_status.config = config;

            return service_status;
        }

        }
```

Б.26 Файл src/device/lib/rpc/relay/src/Server.cpp

```
        #include "open_greenery/rpc/relay/Server.hpp"
        #include <cassert>
        #include <utility>
        #include <grpcpp/server_builder.h>
        #include <spdlog/spdlog.h>
        #include <google/protobuf/empty.pb.h>


        namespace open_greenery::rpc::relay
        {

        grpc::Status Server::Service::SetConfig(
                grpc::ServerContext * context,
                const open_greenery::rpc::relay::Config * request,
                google::protobuf::Empty * response)
        {
            std::ignore = context;
            std::ignore = response;
            open_greenery::dataflow::relay::Config received_config{
                    QTime::fromMSecsSinceStartOfDay(request->day_start()),
                    QTime::fromMSecsSinceStartOfDay(request->day_end())
            };

                    auto   TimeStr   =   [](const   QTime   &   t)   {   return
t.toString("hh:mm:ss").toStdString(); };
            spdlog::debug("Relay config received: day start={}, end={}",
                          TimeStr(received_config.day_start),
                          TimeStr(received_config.day_end));

            m_config_update_handler(received_config);
            return {}; // OK
        }
```

```
grpc::Status Server::Service::ManualControl(
        grpc::ServerContext * context,
        const open_greenery::rpc::relay::ManualControlRequest * request,
        google::protobuf::Empty * response)
{
    std::ignore = context;
    std::ignore = response;

    open_greenery::dataflow::relay::Control control;

    switch (request->control())
    {
        case ManualControlRequest::CONTROL_ENABLE:
            spdlog::debug("Manual enable received");
            control = open_greenery::dataflow::relay::Control::ENABLE;
            break;
        case ManualControlRequest::CONTROL_DISABLE:
            spdlog::debug("Manual disable received");
            control = open_greenery::dataflow::relay::Control::DISABLE;
            break;
        case ManualControlRequest::CONTROL_TOGGLE:
            spdlog::debug("Manual toggle received");
            control = open_greenery::dataflow::relay::Control::TOGGLE;
            break;
        default:
            assert(false && "Unknown relay::Control type");
            // Fictive assignment to avoid warning
            control = open_greenery::dataflow::relay::Control::DISABLE;
    }
    m_manual_control_handler(control);

    return {};
}

grpc::Status Server::Service::SetMode(
        grpc::ServerContext * context,
        const open_greenery::rpc::relay::ModeSetting * request,
        google::protobuf::Empty * response)
{
    std::ignore = context;
    std::ignore = response;

    open_greenery::dataflow::relay::Mode mode;

    switch (request->mode())
    {
        case ModeSetting::MODE_MANUAL:
            spdlog::debug("Manual mode setting received");
            mode = open_greenery::dataflow::relay::Mode::MANUAL;
            break;
        case ModeSetting::MODE_AUTO:
            spdlog::debug("Auto mode setting received");
            mode = open_greenery::dataflow::relay::Mode::AUTO;
            break;
        default:
            assert(false && "relay relay::Mode type");
            // Fictive assignment to avoid warning
            mode = open_greenery::dataflow::relay::Mode::MANUAL;
    }

    m_mode_update_handler(mode);

    return {}; // OK
```

```
    }

    grpc::Status Server::Service::GetRelayStatus(
            grpc::ServerContext * context,
            const google::protobuf::Empty * request,
            open_greenery::rpc::relay::RelayStatus * response)
    {
        std::ignore = context;
        std::ignore = request;

        response->set_is_enabled(m_relay_status_request_handler());
             spdlog::debug("{}  status  sent",  (response->is_enabled()  ?
"Enabled" : "Disabled"));

        return {}; // OK
    }

    grpc::Status Server::Service::GetServiceStatus(
            grpc::ServerContext * context,
            const google::protobuf::Empty * request,
            open_greenery::rpc::relay::ServiceStatus * response)
    {
        std::ignore = context;
        std::ignore = request;

            open_greenery::dataflow::relay::ServiceStatus  service_status  =
m_service_status_request_handler();

        auto * relay_status = new RelayStatus();
        relay_status->set_is_enabled(service_status.relay_enabled);

        auto * config = new Config();
                                                                    config-
>set_day_start(service_status.config.day_start.msecsSinceStartOfDay());
                                                                    config-
>set_day_end(service_status.config.day_end.msecsSinceStartOfDay());

        auto * mode = new ModeSetting();
        mode->set_mode(
                                                service_status.mode    ==
open_greenery::dataflow::relay::Mode::MANUAL ?
                    ModeSetting::MODE_MANUAL : ModeSetting::MODE_AUTO
        );

        response->set_allocated_mode_settings(mode);
        response->set_allocated_relay_status(relay_status);
        response->set_allocated_config(config);

        return {}; // OK
    }

    Server::Server(const std::string & host)
    {
        ::grpc::ServerBuilder builder;
        builder.AddListeningPort(host, ::grpc::InsecureServerCredentials());
        builder.RegisterService(&m_service);
        m_server = builder.BuildAndStart();
    }

    Server::~Server()
    {
        shutdown();
    }
```

```
      void Server::wait()
      {
          assert(m_server && "LightProxyServer wait: server is nullptr");
          m_server->Wait();
      }

      void Server::shutdown()
      {
          assert(m_server && "LightProxyServer shutdown: server is nullptr");
          m_server->Shutdown();
      }

      void Server::onConfigUpdate(
              open_greenery::dataflow::common::AsyncReceive<
                      open_greenery::dataflow::relay::Config> handler)
      {
          m_service.m_config_update_handler = std::move(handler);
      }

      void Server::onManualControl(
              open_greenery::dataflow::common::AsyncReceive
                      <open_greenery::dataflow::relay::Control> handler)
      {
          m_service.m_manual_control_handler = std::move(handler);
      }

      void Server::onModeUpdate(
              open_greenery::dataflow::common::AsyncReceive
                      <open_greenery::dataflow::relay::Mode> handler)
      {
          m_service.m_mode_update_handler = std::move(handler);
      }
      void
Server::onStatusRequest(open_greenery::dataflow::common::AsyncProvide<bool>
handler)
      {
          m_service.m_relay_status_request_handler = std::move(handler);
      }
      void Server::onUpdate(
              open_greenery::dataflow::common::AsyncReceive
                      <open_greenery::dataflow::relay::Config> receive)
      {
          onConfigUpdate(std::move(receive));
      }
      void Server::onUpdate(
              open_greenery::dataflow::common::AsyncReceive
                      <open_greenery::dataflow::relay::Control> receive)
      {
          onManualControl(std::move(receive));
      }
      void Server::onUpdate(
              open_greenery::dataflow::common::AsyncReceive
                      <open_greenery::dataflow::relay::Mode> receive)
      {
          onModeUpdate(std::move(receive));
      }
      void
Server::onRequest(open_greenery::dataflow::common::AsyncProvide<bool>
provide)
      {
          onStatusRequest(std::move(provide));
      }
      void Server::onRequest(
              open_greenery::dataflow::common::AsyncProvide
```

```
                    <open_greenery::dataflow::relay::ServiceStatus> provide)
{
    m_service.m_service_status_request_handler = std::move(provide);
}
}
```

Відомість кваліфікаційної роботи