**Пояснювальна записка**

до курсового проекту

на тему: система контролю веб сервера

Львів 2023

Завдання на курсовий проект

Система контролю веб сервера при розробці Angular проектів.

Проект складається із двох частин:

* Arduino з дисплеєм, джойстиком і двома кнопками яка спілкується з комп'ютером по COM порту.
* Сервіс демон який слухає COM порт і виконує дії в залежності від отриманої комнади.

Arduino повинна відображати список проектів, обирати проект зі списку, відображати статус, надсилати запити до сервіса про запуск проекту, зупинку проекту, перезапуск проекту.

Анотація

Пояснювальна записка складається з 4 розділів, містить 8 рисунків, 41 таблицю та 2 додатки.

Курсовий проект присвячений розробці системи контролю веб сервера.

Метою цієї роботи є підвищення ефективності розробників при розробці Angular проектів.

У першому розділі наведені загальні положення, опис та аналіз предметної області, огляд існуючих рішень, сформульовані функціональні і нефункціональні вимоги, головна мета, цілі та призначення програми.

У другому розділі наведено проектування системи, розробка архітектури програми, алгоритмів її роботи та вибір засобів.

У третьому розділі наведено результати реалізації проектованого рішення та описано розроблені методи та класи.

У четвертому розділі наведено результати тестування програмного забезпечення.

Результатом роботи над проєктом є система що складається із програмної та апаратної частини що дозволяє запускати, зупиняти та перезапускати веб ссервер.

Зміст

[Вступ 5](#_sab7le88mnoy)

[1. Призначення та цілі розробки системи 6](#_5a1emv1aqu2d)

[1.1 Загальні положення 6](#_xkioyr9b0i3r)

[1.2 Змістовний опис і аналіз предметної області 6](#_rr0o0qv8oi6u)

[1.3 Опис ng-deck 7](#_kro7wtlodoon)

[1.4 Мета створення ng-deck 7](#_7yiuhe8t3omm)

[1.5 Очікувані результати 7](#_1o62afw31dsq)

[1.6 Огляд існуючих рішень 8](#_otb9g4w6bw1m)

[1.7 Вимоги до системи 8](#_fzq1xflwlxud)

[1.7.1 Розроблення нефункціональних вимог 9](#_dngtuut1tmp)

[1.7.2 Вимоги до апаратної частини 10](#_xx725afq5bun)

[1.7.3 Вимоги до програмної частини 10](#_n3av2f45qtgc)

[2. Проектування системи 12](#_aebivy9asd6r)

[2.1 Опис процесу діяльності 12](#_y7f1cxh9xu1w)

[2.2 Моделювання та аналіз програмного забезпечення 13](#_y8uz3p9xjvh4)

[2.3 Архітектура програмного забезпечення 14](#_144mhng6anzq)

[2.3.1 Опис архітектури Arduino nano 15](#_29cyqyaqcud7)

[2.3.2 Опис архітектури сервіса 17](#_9bk7fpysqnrh)

[3.Моделювання та конструювання програмного забезпечення 21](#_9q74u0o8zaqa)

[3.1 Детальний опис класів та методів Arduino nano 21](#_9c1b15q7utzt)

[3.2 Детальний опис класів та методів сервіса 27](#_nyd749l8rlr9)

[4. Аналіз якості та тестування програмного забезпечення 35](#_pfvtcagx3q33)

[4.1 Аналіз якості ПЗ 35](#_vrwag22ewpy7)

[4.2 Опис процесів тестування 36](#_b8m6ae7hkcta)

[4.3 Опис тест кейсів 37](#_x8qul9ff2ixf)

[4.3.1 unit тести 37](#_p304s6if3uy3)

[4.3.2 е2е тести 37](#_7sjwrrq16lbp)

[Висновки 38](#_fosj084syzez)

[Список використаних джерел 39](#_ykmtomwxyfqy)

# 

# 

# Вступ

Сучасні засоби для розробки програмного забезпечення доволі потужні та зручні для використання розробниками. Однак через свою складність і велику кількість залежностей виникають некритичні проблеми які важко вирішити через те що різні частини часто розробляються різними командами. Найчастіше все вирішується перезапуском проекту. Також зі збільшенням складності програмного забезпечення доводиться робити рутинні дії для того щоб запустити проект.

Рішенням цієї проблеми може стати пристрій який налаштовується один раз, після чого дозволяє спростити рутинні дії та пришвидшити розробку. Такий пристрій повинен стежити за статусом програмного забезпечення та виконувати дії при взаємодії користувача для запуску, зупинки чи перезапуску проекту.

Існуючі рішення які можуть вирішити цю проблему мають ряд недоліків, зокрема високу ціну та малий функціонал, оскільки не призначені для розробників.

Завданням створення даної програми є програмне та апаратне забезпечення, що зможе стежити за станом проекту та забере на себе рутинні дії.

Практичним застосуванням такої програми є допомога у створення веб застосунків.

# 1. Призначення та цілі розробки системи

## 1.1 Загальні положення

З ростом потужності обчислювальної техніки веб застосунки перейшли від шаблонізаторів які працювали на стороні сервера і видавали користувачу готові сторінки до систем які працюють на комп'ютері користувача і перебудовують сторінку в режимі реального часу без необхідності очікувати на відповідь від сервера з новою сторінкою. Така зміна призвела до появи складних фреймворків які полегшують розробку і беруть на себе роботу по перебудові сторінки. Це призвело до зростання складності розробки і виникнення однотипної роботи по запуску, зупинці і перезапуску проектів та виникнення некритичних помилок які важко виправити. Наразі такі фреймворки дуже популярні і швидко розвиваються, хоча деякі проблеми так і залишаються. Такі системи використовують власний сервер для розробки та відлагодження, який оновлюється при змінах файлів проекту.

## 1.2 Змістовний опис і аналіз предметної області

Angular - один із веб фреймворків який використовує Webpack development server. Він стежить за файлами і сам заміняє їх на сервері і оновлює сторінку. Це дуже спрощує розробку, оскільки сервер завжди віддає актуальний проект в режимі реального часу без необхідності перезавантажувати сторінку чи перезапускати сервер. Він запускається із кореня папки проекту і працює поки його не буде зупинено.

Webpack - це збірщик, який збирає усі файли проекту в один формат для передачі серверу. Сам проект може складатися із тисяч файлів, оскільки розбиття проектів на модулі які можна використати повторно спрощує розробку і полегшує читання і відлагодження проекту, а також дозволяє протестувати кожен модуль окремо. Однак для сервера це неважливо і навпаки лише сповільнює виконання, оскільки виникає багато залежностей між файлами. Тому webpack збирає усі файли у кілька модулів для зменшення їх розміру та збільшення швидкості.

## 1.3 Опис ng-deck

Angular використовує webpack dev server, який запускається при розробці, і при зміні файлів проекту він автоматично підхоплює зміни і на гарячу оновлює веб сервер. Наприклад така помилка як 'XXX is not a known element' означає що такого елемента не існує, або неправильно вказані імпорти / експорти, хоча якщо все добре прописано, але ця помилка появляється це вже дивно(таке стається досить часто при створенні нових файлів, і є добре відомою проблемою Angular) . Єдиним рішенням є зайти в термінал, зупинити сервер і знову його запустити щоб він правильно підхопив усі файли для збірки.

## 1.4 Мета створення ng-deck

Метою створення системи є підвищення комфорту під час розробки та досягнення наступних цілей:

* Автоматизація рутинних задач розробника
* Підвищення швидкості розробки
* Подолання проблеми неправильного підхоплення нових файлів

## 1.5 Очікувані результати

Результатом має стати система, яка:

* Взаємодіє з комп'ютером по СОМ порту
* Шукає Angular проекти у вказаних директоріях
* Відображає список проектів на дисплеї Arduino
* Дозволяє вибрати проект зі списку(активний проект)
* Дозволяє запустити активний проект з Arduino
* Дозволяє зупинити активний проект з Arduino
* Дозволяє перезапустити активний проект з Arduino
* Відображає додаткову інформацію про активний проект(статус, порт)
* Має просте налаштування

## 1.6 Огляд існуючих рішень

Єдиним готовим засобом який може частково виконувати функції ng-deck є stream deck - клавіатура з програмованими кнопками. Цей пристрій призначений для ведення прямих трансляцій і дозволяє налаштувати різні дії на натискання кнопок. Його можна би було налаштувати для запуску Angular проекту по натисканні кнопки, але він не може відстежувати стан та перезапускати проект, оскільки не має такого функціоналу. Також серед його недоліків - прив'язаність до Windows та висока ціна(від 400$).

## 1.7 Вимоги до системи

Система повинна складатися із двох частин

* Arduino з дисплеєм, енкодером і двома кнопками яка спілкується з комп'ютером по COM порту.
* Сервіс демон який слухає COM порт і виконує дії в залежності від отриманої комнади.

Діаграма варіантів використання зображена на рисунку 1.1. У таблиці 1.1 наведено функціональні вимоги.

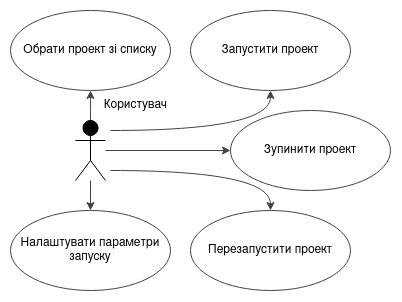


Рисунок 1.1 Схема варіантів використання

Таблиця 1.1 - Функціональні вимоги

| Варіант використання | Функціональна вимога |
| --- | --- |
| Обрати проект зі списку | Система має надавати можливість обрати проект зі списку на дисплеї |
| Запустити проект | Система повинна по натисканні кнопки запускати обраний проект |
| Зупинити проект | Система повинна по натисканні кнопки, якщо проект запущено,зупиняти обраний проект |
| Перезапустити проект | Система повинна по натисканні кнопки, якщо проект запущено, перезапускати обраний проект |
| Налаштувати параметри запуску | Система повинна надавати можливість міняти параметри запуску для проектів та налаштовувати аргументи які будуть використовуватися |

### 1.7.1 Розроблення нефункціональних вимог

Було виділено наступні нефункціональні вимоги

* продукт повинен працювати на операційній системі GNU/Linux, зокрема на дистрибутиві Arch Linux.
* продукт повинен підтримувати Angular 9 і вище
* має бути передбачений захист від некоректний дій користувача
* інтерфейс має бути зрозумілим та простим для використання
* просте налаштування
* висока швидкодія

### 1.7.2 Вимоги до апаратної частини

В основі апаратної частини Arduino nano, які підключається до комп'ютера по СОМ порту та отримує з нього інформацію та передає команди. Arduino повинна відображати список проектів, обирати проект зі списку, відображати статус, надсилати запити до сервіса про запуск проекту, зупинку проекту, перезапуск проекту. Також апаратна частина повинна мати 2 кнопки та енкодер.

Список проектів повинен відображатися на рідкокристалічному дисплеї 1602. За допомогою джойстика відбувається навігація по списку (вниз - вниз по списку, вгору - вгору по списку). По натисканні на другу кнопку вибраний проект стає активним і може бути запущений натисканням першої кнопки. Друге натискання тієї ж кнопки повинно перезавантажити проект, про що необхідно повідомити на рідкокристалічному дисплеї. Друга кнопка повинна зупиняти активний проект

### 1.7.3 Вимоги до програмної частини

Програмна частина проекту це сервіс який слухає СОМ порт та виконує дії в залежності від отриманих команд чи подій. Програма повинна мати файл конфігурації, який дозволяє вказати директорії, в яких система повинна шукати Angular проекти та можливість додати аргументи для запуску. Сервіс повинен розрізняти валідні Angular проекти поміж інших файлів та директорій.

При отриманні команди від мікроконтролера про запуск проекту система повинна запускати обраний проект у новому потоці для можливості подальшої комунікації з Arduino.

Сервіс повинен зберігати стан апаратної частини на випадок втрати живлення мікроконтролера або випадкового від'єднання кабеля.

# 2. Проектування системи

## 2.1 Опис процесу діяльності

Першим етапом роботи системи є сканування вказаних папок на комп'ютері та пошук серед них Angular проектів. Паралельно з цим йде пошук порта до якого під'єднано пристрій і встановлення з'єднання.

Після цього відбувається передача цих проектів Arduino для відображення їх на дисплеї.

Коли список відображено на дисплеї по ньому можна переміщатися і обрати потрібний проект

Після того як проект було обрано його можна запустити

Запущений проект можна перезапустити або зупинити

На стороні комп'ютера є процес який стежить за зміною папок з проектами, і якщо проект буде додано чи видалено то Arduino дізнається про це і оновить свій список.

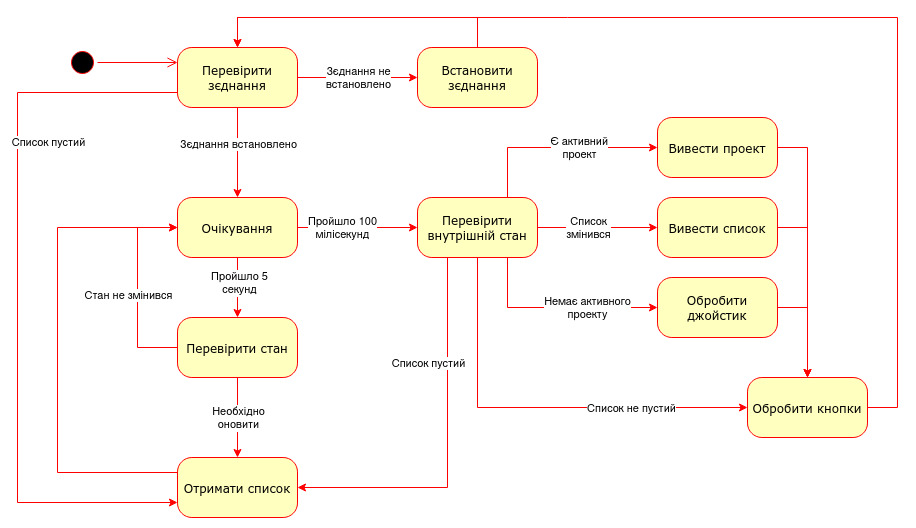


Рисунок 2.1 - Діаграма станів Arduino

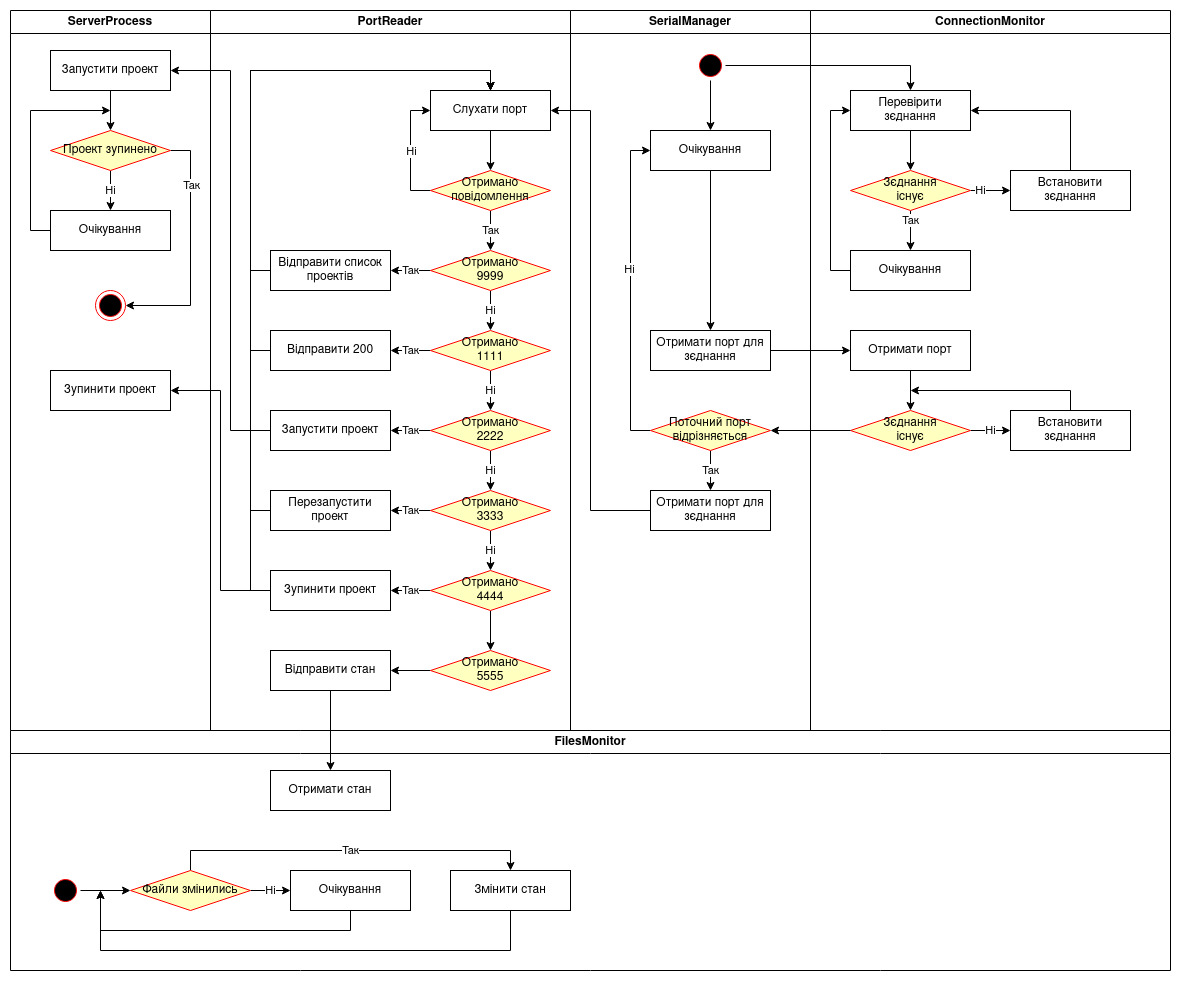


Рисунок 2.2 - Діаграма станів сервіса

## 2.2 Моделювання та аналіз програмного забезпечення

Загальні процеси, які проходить користувач, включають процес вибору проекту, запуску проекту, зупинки проекту та перезапуску проекту.

Послідовний опис етапу вибору проекту:

* Після успішного встановлення з'єднання з сервісом та отримання списку проектів на дисплеї буде відображено список проектів із вказівником на верхній проект
* Використання джойстика для переміщення по списку(джойстик вгору - список переміщується вгору, джойстик вниз - список переміщається вниз)
* Натискання другої кнопки щоб встановити проект навпроти якого стоїть вказівник як активного

На етапі запуску проекту користувач натискає на першу кнопку. Якщо вибрано активний проект то буде відправлено команду сервісу про запуск проекту.

На етапі зупинки проекту користувач натискає на другу кнопку. Якщо вибрано активний проект то буде відправлено команду сервісу про зупинку проекту.

На етапі перезапуску проекту користувач натискає на першу кнопку. Якщо вибрано активний проект то буде відправлено команду сервісу про перезапуск проекту.

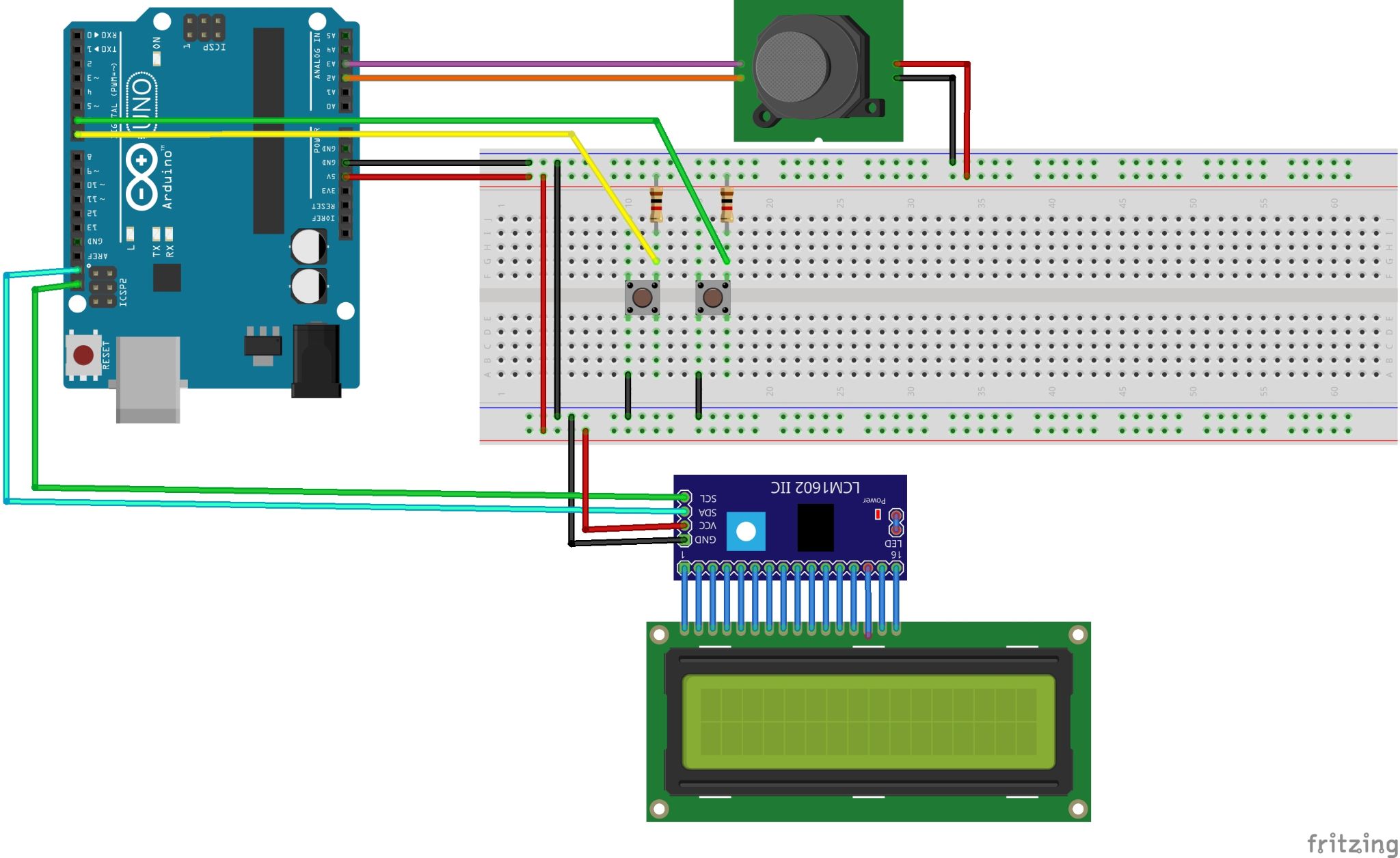


Рисунок 2.3 Схема Arduino

## 2.3 Архітектура програмного забезпечення

Програмне забезпечення складається із двох частин:

* Arduino nano яка відправляє команди по COM порту
* Сервіс який слухає COM порт та виконує дії в залежності від отриманої команди

Для написання прошивки для Arduino nano було використано мову програмування C++ з бібліотекою від Arduino. Для роботи з Arduino буде використано PlatformIO що є розширенням для інтегрованих середовищ розробки. PlatformIO значно спрощує процес розробки і має гнучкі налаштування, а завдяки інтеграції з іншими середовищами для розробки не потрібно опановувати нові програми.

Для написання сервіса буде використано мову програмування Java. Для організації взаємодії по COM порту на стороні сервіса буде використано бібліотеку jssc яка є обгорткою над методами для роботи з COM портом самої Java. Також було обрано Lombok для генерації рутинного коду і бібліотеку Gson для читання та запису конфігураційного файлу у форматі JSON. Для тестування буде використано бібліотеку Junit5 i Mockito для створення фальшивих обєктів при юніт тестуванні, оскільки цей набір є стандартом для тестування Java програм і не має аналогів. Для логування було обрано бібліотеку Log4j2 оскільки вона проста у використанні, підтримується Lombok і має гнучкі налаштування.

Схема взаємодії модулів зображена на рисунку 2.2

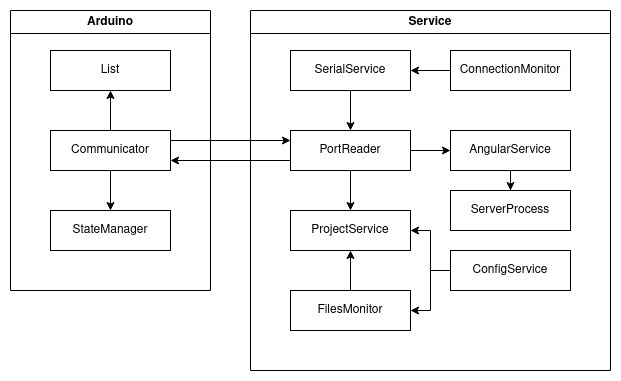


Рисунок 2.4 Схема взаємодії модулів

### 2.3.1 Опис архітектури Arduino nano

Прошивка для Arduino містить 7 класів. Діаграма класів зображена на рисунку 2.3. У таблиці 2.1 наведено опис класів та їх призначення

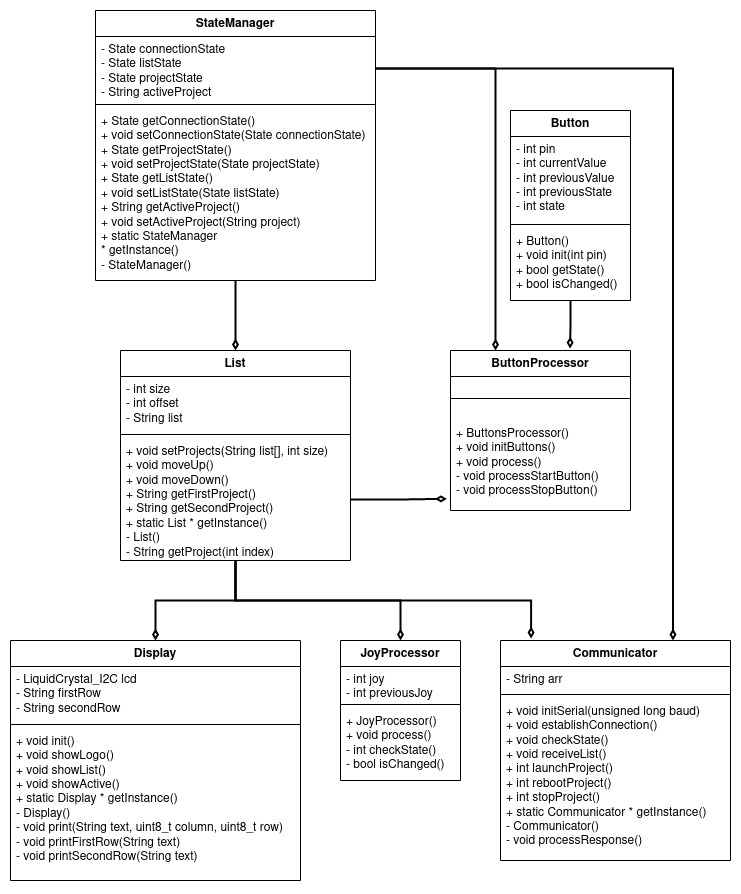


Рисунок 2.5 Діаграма класів

Таблиця 2.1 – Опис класів та їх призначення

| Клас | Опис та призначення класу |
| --- | --- |
| StateManager | Клас що зберігає стан програми, використовується щоб контролювати і міняти стан. Основний клас що використовується по всій програмі для отримання поточного стану з'єднання / списку / проекту. Зберігає назву вибраного проекту |
| List | Клас що зберігає масив проектів і дозволяє отримати проекти для виводу на екран. Реалізує функціонал для переміщення по списку |
| Display | Обгортка над внутрішнім класом для виводу інформації на дисплей. Зберігає поточну інформацію яка виведена на дисплеї і виводить нову тільки якщо вона відрізняється від попередньої для запобігання надлишковому морганню. |
| Communicator | Клас який спілкується із сервісом, надсилає команди і обробляє відповідь. Потрібен для зв'язку із сервісом. |
| JoyProcessor | Клас що обробляє джойстик, зміну його стану та міняє список якщо джойстик переміщується |
| ButtonProcessor | Клас що зберігає дві кнопки та обробляє зміну їх стану і відправляє відповідні команди |
| Button | Клас що обробляє натискання кнопок, відстежує попередній та поточний стан для зручного використання в інших місцях. Потрібен щоб правильно обробляти натискання і запобігати некоректним діям. |

### 2.3.2 Опис архітектури сервіса

Сервіс складається із чотирьох пакетів - config, data, models та services. У свою чергу пакет services містить 3 пакети - angular, projects та serial. У таблиці 2.2 описано кожен пакет та класи що входять до них. Архітектура сервіса зображена на рисунку 2.6. Діаграма класів зображена на рисунку 2.7.

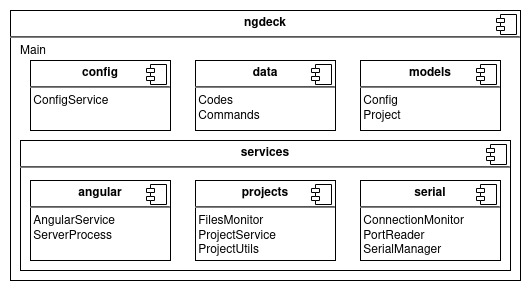


Рисунок 2.6 Архітектура сервіса

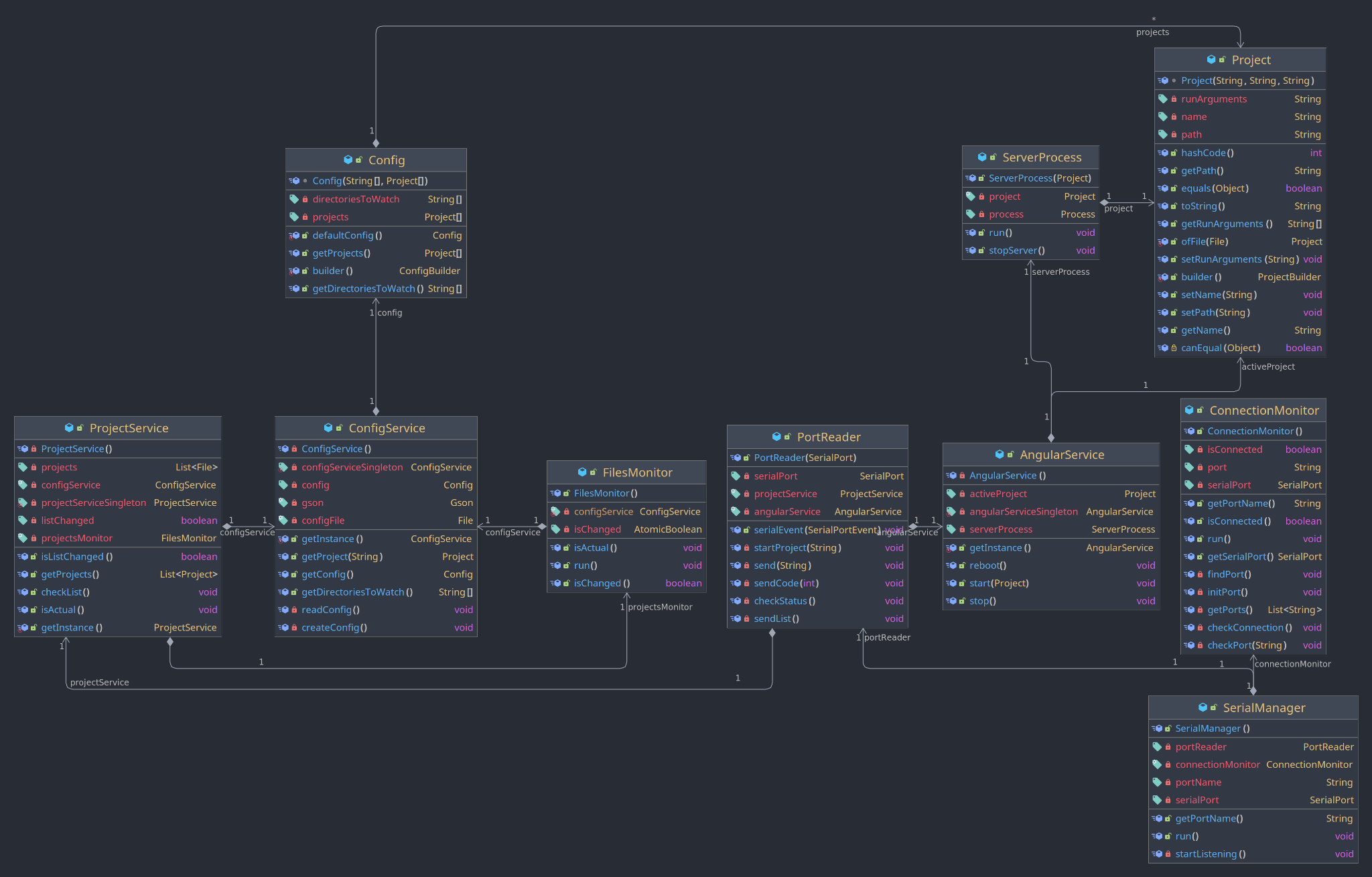


Рисунок 2.7 Діаграма класів сервіса

Таблиця 2.2 – Опис пакетів та класів сервіса

| Назва пакету | |
| --- | --- |
| config | |
| Клас | Опис та призначення класу |
| ConfigService | Клас що відповідає за читання і при потребі створення конфігураційного файлу. |
| Назва пакету | |
| data | |
| Клас | Опис та призначення класу |
| Codes | Клас що містить константи із кодами для комунікації. Використовується для уніфікації повідомлень при обміні даних між пристроями |
| Commands | Клас що містить команди для комунікації. Використовується для уніфікації повідомлень при обміні даних між пристроями |
| Назва пакету | |
| models | |
| Клас | Опис та призначення класу |
| Config | Клас модель який використовується для зчитування даних із конфігураційного файлу. Потрібен для зчитування JSON файлів. |
| Project | Клас модель що використовується для зчитування даних із конфігураційного файлу та іншими класами як обгортка над проектами |
| Назва пакету | |
| services - angular | |
| Клас | Опис та призначення класу |
| AngularService | Клас що контролює Angular проекти, зберігає активний проект, запускає, зупиняє і перезавантажує їх. |
| ServerProcess | Клас що запускає Angular проект у новому потоці, використовується і контролюється класом AngularService |
| services - projects | |
| FilesMonitor | Клас що стежить за змінами файлів у вказаних директоріях |
| ProjectService | Клас що зберігає та повертає актуальний список проектів які є на комп'ютері |
| ProjectUtils | Допоміжний клас який забезпечує функції для пошуку та виявлення Angular проектів та перетворення у потрібний формат. |
| services - serial | |
| SerialService | Клас що реалізує функціонал для роботи з COM портом |
| ConnectionMonitor | Клас що стежить за поточним з'єднанням та шукає порти для встановлення з'єднання |
| PortReader | Клас що запускається у новому потоці і слухає COM порт та виконує дії в залежності від отриманих команд |

# 

# 3.Моделювання та конструювання програмного забезпечення

## 3.1 Детальний опис класів та методів Arduino nano

**StateManager** - Клас що зберігає стан з'єднання, списку і проекту а також поточний активний проект. Реалізує паттерн Singleton.

Клас містить 3 приватні поля типу State та одне приватне поле String:

* State connectionState
* State listState
* State projectState
* String activeProject

Клас містить гетери і сетери для всіх полів. Конструктор класу є приватним, щоб створити клас можна було лише за допомогою публічного статичного методу getInstance, який повертає єдиний об'єкт класу.

**List** - клас що зберігає масив проектів і реалізує меню для переміщення по ньому. Реалізує паттерн Singleton. Клас містить чотири приватні поля:

* int size - розмір масиву
* int offset - відступ від початку масиву для виведення на екран
* String list[] - масив проектів
* StateManager stateManager - посилання на об'єкт StateManager

Методи та їх призначення описані у таблиці 3.1

Таблиця 3.1 – Опис методів класу List

| Метод | Призначення |
| --- | --- |
| void setProjects(String list[], int size) | Публічний сеттер що отримує масив проектів і його розмір |
| void moveUp() | Публічний метод що зменшує відступ від початку масиву для реалізації переміщення по списку |
| void moveDown() | Публічний метод що збільшує відступ від початку масиву для реалізації переміщення по списку |
| String getFirstProject() | Публічний метод що повертає перший проект враховуючи відступ для відображення на дисплеї |
| String getSecondProject() | Публічний метод що повертає другий проект враховуючи відступ для відображення на дисплеї |
| static List \* getInstance() | Публічний метод що повертає об'єкт класу List, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| List() | Приватний конструктор. Потрібен для реалізації патерну Singleton |
| String getProject(int index) | Приватний метод для отримання об'єкта масиву за індексом. Повертає пустий рядок якщо масив пустий |

**Display** - Класс обгортка над классом LiquidCrystal\_I2C. Зберігає поточний стан дисплея, реалізує функції для виводу на дисплей логотипа, списку і активного проекту. Запобігає надлишковому морганню дисплея завдяки заміні тексту лише коли він відрізняється від того що виведено. Клас містить 5 приватних полів.

* LiquidCrystal\_I2C lcd - клас що працює з дисплеєм на шині I2C
* List \*list - об'єкт класу List
* StateManager \*stateManager - обєкт класу StateManager
* String firstRow - перший рядок що зараз виведено на дисплей
* String secondRow - другий рядок що зараз виведений на дисплей

Методи та їх призначення описані у таблиці 3.2

Таблиця 3.2 – Опис методів класу Display

| Метод | Призначення |
| --- | --- |
| void init() | Публічний метод який ініціалізує дисплей і встановлює з'єднання по шині I2C |
| void showLogo() | Публічний метод який виводить логотип ng-deck на дисплей |
| void showList() | Публічний метод який виводить список проектів на дисплей |
| void showActive() | Публічний метод який виводить активний проект на дисплей |
| static Display \* getInstance() | Публічний метод що повертає об'єкт класу Display, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| Display() | Приватний конструктор. Потрібен для реалізації патерну Singleton |
| void print(String text, uint8\_t column, uint8\_t row) | Приватний метод який виводить текст починаючи з вказаного рядку і стовпця |
| void printFirstRow(String text) | Приватний метод який виводить текст на перший рядок дисплею. Виводить на дисплей лише в тому випадку якщо переданий текст відрізняється від того що вже виведено |
| void printSecondRow(String text) | Приватний метод який виводить текст на другий рядок дисплею. Виводить на дисплей лише в тому випадку якщо переданий текст відрізняється від того що вже виведено |

**Communicator** - Клас що реалізує методи для взаємодії із сервісом по COM порту. Реалізує паттерн Singleton. Містить три приватні поля:

* List list - посилання на об'єкт List
* StateManager stateManager - посилання на об'єкт StateManager
* String[] arr - масив, в який записується відповідь сервіса при отриманні масиву проектів

Методи та їх призначення описані у таблиці 3.3

Таблиця 3.3 – Опис методів класу Communicator

| Метод | Призначення |
| --- | --- |
| void initSerial(unsigned long baud) | Публічний метод для ініціалізації і запуску комунікацію по COM порту зі вказаною швидкістю. |
| void establishConnection() | Публічний метод для встановлення з'єднанням із сервісом |
| void checkState() | Публічний метод що відправляє команду сервісу для перевірки статусу списку |
| void receiveList() | Публічний метод що відправляє команду сервісу для отримання списку |
| int launchProject() | Публічний метод що відправляє команду сервісу про запуск вибраного проекту |
| int rebootProject() | Публічний метод що відправляє команду сервісу про перезапуск проекту |
| int stopProject() | Публічний метод що відправляє команду сервісу про зупинку проекту |
| static Communicator \* getInstance() | Публічний метод що повертає об'єкт класу Communicator, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| Communicator() | Приватний конструктор. Потрібен для реалізації патерну Singleton |
| void processResponse() | Приватний метод для обробки відповіді сервіса. Використовується для обробки кодів які відправляє сервіс |

**JoyProcessor** - Клас що обробляє переміщення джойстика. Зберігає поточний і попередній стан порту до якого підключено джойстик для визначення напрямку куди було переміщено джойстик. Містить три приватні поля:

* int joy - поточний напрямок джойстика
* int previousJoy - попередній напрямок джойстика
* List list - посилання на об'єкт List

Методи та їх призначення описані у таблиці 3.4

Таблиця 3.4 – Опис методів класу Communicator

| Метод | Призначення |
| --- | --- |
| JoyProcessor() | Публічний конструктор |
| void process() | Публічний метод який при виклику визначає чи змінився стан джойстика і виконує відповідні дії |
| int checkState() | Приватний метод який визначає поточний стан |
| bool isChanged() | Приватний метод який перевіряє чи стан джойстика змінився |

**ButtonProcessor** - клас що зберігає дві кнопки та обробляє зміну їх стану і виконує відповідні дії по встановленню активного проекту, запуску, зупинці та перезавантаженні проекту. Клас містить 5 приватних полів:

* StateManager stateManager - посилання на об'єкт StateManager
* Communicator communicator - посилання на об'єкт Communicator
* List list - посилання на об'єкт List
* Button startButton - посилання на об'єкт Button
* Button stopButton - посилання на об'єкт Button

Методи та їх призначення описані у таблиці 3.5

Таблиця 3.5 – Опис методів класу Communicator

| Метод | Призначення |
| --- | --- |
| ButtonsProcessor() | Публічний клас |
| void initButtons() | Публічний метод який ініціалізує порти до яких підключено кнопки |
| void process() | Публічний метод який при виклику перевіряє стан кнопок та виконує відповідні методи |
| void processStartButton() | Приватний метод який викликається методом process() і обробляє натискання першої кнопки |
| void processStopButton() | Приватний метод який викликається методом process() і обробляє натискання другої кнопки |

**Button** - клас обгортка над портом до якого підключено кнопку для визначення стану і відстеження змін. Клас містить 5 приватних полів:

* int pin - номер піна до якого підключено кнопку
* int currentValue - поточне значення порту кнопки
* int PreviousValue - попереднє значення порту кнопки
* int state - поточний стан кнопки
* int PreviousState - попередній стан кнопки

Методи та їх призначення описані у таблиці 3.6

Таблиця 3.6 – Опис методів класу Button

| Метод | Призначення |
| --- | --- |
| Button() | Публічний конструктор |
| void init(int pin) | Публічний метод який ініціалізує порт до якого підключено кнопку |
| bool getState() | Публічний метод який визначає поточний стан порту до якого підключено кнопку |
| bool isChanged() | Публічний метод який визначає чи змінився стан кнопки |

### 

## 3.2 Детальний опис класів та методів сервіса

**Пакет config**

**ConfigService -** клас який призначений для читання та створення конфігураційного файлу якщо його не існує або він невалідний. Реалізує паттерн Singleton Використовує бібліотеку gson для читання та запису JSON файлів. Створює файл у домашній директорії у прихованій папці .ng-deck з назвою config.json, за цим же шляхом шукає його. Клас містить 3 приватні поля

* Config config - об'єкт що зчитується із файлу
* Gson gson - об'єкт класу Gson для читання та запису JSON файлів
* File configFile - шлях до конфігураційного файлу

Методи та їх призначення описані у таблиці 3.7

Таблиця 3.7 – Опис методів класу ConfigService

| Метод | Призначення |
| --- | --- |
| ConfigService() | Перевизначення стандартного конструктора щоб зробити його приватним |
| ConfigService getInstance() | Публічний метод що повертає об'єкт класу ConfigService, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| Project getProjects(String) | Публічний метод який повертає об'єкт Project із вказаним ім'ям якщо він вказаний у конфігураційному файлі, в іншому випадку повертає null |
| Config getConfig() | Публічнйи метод який повертає поточний конфігураційний файл |
| String[] getDirectoriesToWatch() | Публічний метод який повертає масив директорій які вказані у конфігураційному файлі |
| void readConfig() | Приватний метод який читає конфігураційний файл із файлової системи |
| void createConfig() | Приватний метод який створює конфігураційний файл за замовчуванням у файловій системі |

**Пакет data**

**Codes -** Клас що містить константами із кодами для комунікації між Arduino та сервісом. Використовуються для відправки як відповідь на отримане повідомлення щоб переконатися що повідомлення отримано чи сталася якась помилка.

**Commands -** Клас що містить константи із кодами для комунікації між Arduino та сервісом. Використовується для порівняння отриманого повідомлення із командами для визначення дії яку потрібно виконати.

**Пакет models**

**Config** - Клас модель який використовується як шаблон для читання і запису бібліотекою Gson. Гетери і Сетери згенеровані бібліотекою lombok. Реалізує паттерн Builder завдяки бібліотеці lombok. Містить 2 приватні поля:

* String[] directoriesToWatch - директорії в яких потрібно шукати проекти
* Project[] projects - налаштування проектів

Методи та їх призначення описані у таблиці 3.8

Таблиця 3.8 – Опис методів класу Config

| Метод | Призначення |
| --- | --- |
| Config defaultConfig() | Публічний статичний метод який повертає стандартну конфігурацію. Використовується для створення конфігураційного файлу або коли конфігураційний файл недоступний. |

**Project -** Клас модель який є обгорткою над проектами. Потрібен для зручного доступу до проектів і маніпуляції ними. Гетери і Сетери згенеровані бібліотекою lombok. Реалізує паттерн Builder завдяки бібліотеці lombok. Містить 3 приватні поля:

* String name - назва проекту
* String path - шлях до проекту
* String runArguments - аргументи для запуску проекту

Методи та їх призначення описані у таблиці 3.9

Таблиця 3.9 – Опис методів класу Project

| Метод | Призначення |
| --- | --- |
| Project ofFile(File file) | Публічний статичний метод який створює об'єкт класу Project із Fike |
| String[] getRunArguments | Повертає аргументи для запуску проекту у форму масиву для використання при запуску процесу |

**Пакет services - angular**

**AngularService -** Клас який виконує дії над проектами. Запускає, зупиняє і перезавантажує проект. Зберігає поточний запущений проект. Проекти запускаються у новому потоці у класі ServerProcess. Реалізує паттерн Singleton. Містить 2 приватні поля:

* Project activeProject - поточний активний проект
* ServerProcess serverProcess - процес в якому запущено Angular проект

Методи та їх призначення описані у таблиці 3.10

Таблиця 3.10 – Опис методів класу AngularService

| Метод | Призначення |
| --- | --- |
| AngularService() | Перевизначення стандартного конструктора щоб зробити його приватним |
| AngularService getInstance() | Публічний метод що повертає об'єкт класу AngularService, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| void reboot() | Публічний метод для перезапуску проекту |
| void start(Project) | Публічний метод для запуску проекту. Отримує проект який необхідно запустити |
| void stop() | Публічний метод для зупинки проекту |

**ServerProcess** - Клас що запускає Angular проект у новому потоці. Наслідує клас Thread самої java. Містить два приватні поля:

* Project project - активний проект
* Process process - процес який запущено

Методи та їх призначення описані у таблиці 3.11

Таблиця 3.11 – Опис методів класу ServerProcess

| Метод | Призначення |
| --- | --- |
| ServerProcess(Project) | Публічний конструктор який отримує проект який необхідно запустити у новому потоці |
| void run() | Запускає проект у новому потоці |
| void stopServer() | Зупиняє процес який було запущено |

**Пакет services - projects**

**FilesMonitor -** Клас який стежить за змінами файлів у директоріях які вказані у конфігураційних файлах. Реагує на системні хуки по створенню та видаленні файлів. Працює у новому потоці. Містить два приватні поля:

* ConfigService - посилання на конфігураційний сервіс
* AtomicBoolean isChanged - змінна типу boolean яка вказує чи змінились файли на диску. Використовує атомарні операції для зміни стану.

Методи та їх призначення описані у таблиці 2.12

Таблиця 2.12 – Опис методів класу FilesMonitor

| Метод | Призначення |
| --- | --- |
| FilesMonitor() | Публічний конструктор який створює об'єкт класу. |
| void run() | Запускає у новому потоці процес спостереження за змінами у файлах |
| boolean isChanged() | Повертає чи були змінені файли |
| void isActual() | Викликається коли зміни були опрацьовані іншим класом. |

**ProjectService -** Сервіс для роботи із проектами. Зберігає поточний список проектів і оновлює його при змінах файлів. Реалізує паттерн Singleton.

Містить чотири приватні поля:

* FilesMonitor projectsMonitor - потік що стежить за змінами файлів у директорії
* ConfigService - посилання на конфігураційний сервіс
* List<File> projects - масив файлів що є Angular проектами
* boolean listChanged - чи змінився список, використовується щоб відправити Arduino по запиті

Методи та їх призначення описані у таблиці 3.13

Таблиця 3.13 – Опис методів класу FilesMonitor

| Метод | Призначення |
| --- | --- |
| ProjectService() | Перевизначення стандартного конструктора щоб зробити його приватним |
| ProjectService getInstance() | Публічний метод що повертає об'єкт класу ProjectService, єдиний спосіб отримати об'єкт класу. Потрібен для реалізації патерну Singleton |
| boolean isListChanged() | Повертає чи змінився список |
| void isActual() | Встановлює що зміни були прийняті |
| void checkList() | Опрацьовує зміни якщо FilesMonitor їх виявив. Перевіряє чи список проектів на диску змінився від поточного |
| List<Project> getProjects() | Повертає список проектів |

**ProjectUtils -** Клас що реалізує допоміжний функціонал для роботи ProjectService. Містить функції по пошуку Angular проектів та перетворення їх в інший формат.

Методи та їх призначення описані у таблиці 3.14

Таблиця 3.14 – Опис методів класу FilesMonitor

| Метод | Призначення |
| --- | --- |
| String formatToSend(List<Project> projects) | Публічний статичний метод який форматує список проектів у повідомлення для відправки по послідовному порту |
| ArrayList<File> getProjects(String[] directories) | Публічний статичний метод який повертає масив проектів у вказаних директоріях |
| ArrayList<File> findProjects(String path) | Публічний статичний метод який повертає список проектів за вказаним шляхом. |
| boolean isAngular(File file) | Публічний статичний метод який перевіряє чи вказаний файл є валідним Angular проектом. |

**Пакет services - serial**

**SerialManager -** Клас що керує портами, запускає PortReader при отриманні нового порту від ConnectionMonitor. Містить 4 приватні поля:

* String portName - назва порту який слухається
* SerialPort serialPort - поточний порт
* PortReader portReader - об'єкт класу PortReader який слухає поточний порт
* ConnectionMonitor connectionMonitor - клас що стежить за поточним з'єднанням

Методи та їх призначення описані у таблиці 3.15

Таблиця 3.15 – Опис методів класу SerialManager

| Метод | Призначення |
| --- | --- |
| SerialManager() | Публічний конструктор який створює об'єкт класу. |
| void run() | Перевизначення методу класу Thread який запускається у новому потоці. Стежить за зміною порту і запускає новий PortReader при зміні порту. |
| void startListening() | Запускає новий PortReader. Починає слухати порт. |

**ConnectionMonitor -** Клас що шукає та встановлює з'єднання на порту. Стежить за з'єднанням і шукає новий порт при втраті. Містить три приватні поля:

* String port - назва поточного порту який є активним
* SerialPort serialPort - Поточний активний порт
* boolean isConnected - чи встановлено з'єднання

Методи та їх призначення описані у таблиці 3.16

Таблиця 3.16 – Опис методів класу ConnectionMonitor

| Метод | Призначення |
| --- | --- |
| void run() | Перевизначення методу класу Thread який запускається у новому потоці. Перевіряє чи поточний порт активний і шукає порт якщо з'єднання відсутнє. |
| SerialPort getSerialPort() | Повертає поточний порт, якщо з'єднання не встановлено то чекає на встановлення з'єднання і тоді повертає порт. |
| String getPortName() | Повертає назву поточного порту |
| void findPort() | Шукає порт до якого підключено Arduino |
| void checkConnection() | Приватний метод який перевіряє чи поточне з'єднання є активним |
| void checkPort(String port) | Приватний метод який перевіряє наданий порт, чи підключено до нього Arduino |
| void initPort() | Приватний метод який ініціалізує поточний порт, встановлює режим спілкування, маскуючі символи та інші налаштування для коректної роботи. |
| List<String> getPorts() | Приватний метод який повертає список всіх портів які зараз доступні |

**PortReader -** Клас що слухає COM порт та виконує дії в залежності від отриманої команди. Наслідує клас SerialPortEventListener. Містить три приватні поля:

* SerialPort serialPort - порт по якому ведеться комунікація
* ProjectService projectService - об'єкт класу ProjectService
* AngularService angularService - об'єкт класу AngularService

Методи та їх призначення описані у таблиці 3.17

Таблиця 3.17 – Опис методів класу PortReader

| Метод | Призначення |
| --- | --- |
| PortReader(SerialPort serialPort) | Публічний конструктор який створює об'єкт класу PortReader який буде слухати вказаний порт. |
| void SerialEvent(SerialPortEvent event) | Метод що обробляє запити що відправляє Arduino. Перевизначає метод класу SerialPortEventListener, працює в новому потоці і обробляє Event які виникають. |
| void checkStatus() | Приватний метод що викликається коли Arduino відправляє команду про перевірку стану. |
| void startProject() | Приватний метод що викликається коли Arduino відправляє команду про запуск проекту |
| void sendList() | Приватний метод що викликається коли Arduino відправляє команду про отримання списку. Відправляє розмір а потім назви проектів. |
| void sendCode(int code) | Відправляє код по послідовному порту |
| void send(String message) | Відправляє текст по послідовному порту. |

# 

# 4. Аналіз якості та тестування програмного забезпечення

## 4.1 Аналіз якості ПЗ

Тестування програмного забезпечення є необхідною частиною розробки продукту. За допомогою послідовного виконання тестів можна перевірити функціональну поведінку програми, зручність використання та правильність роботи функціоналу.

Перевагами тестування ПЗ є:

- знаходження та виправлення помилок до того, як продуктом почне

користуватись аудиторія;

- мінімізація технічних втрат та швидкий розвиток процесу розробки

через виконання тестування на декількох стадіях створення;

- оптимізація коду.

Оскільки число можливих тестів навіть для нескладних програмних компонентів практично нескінченне, тому стратегія тестування полягає в тому, щоби провести всі можливі тести з урахуванням наявного часу та ресурсів. Як результат програмне забезпечення (ПЗ) тестують стандартним виконанням програми з метою виявлення багів (помилок або інших дефектів). Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість ПЗ, ризики відмови.

Тестування можна проводити, як тільки створено виконуваний код (навіть частково завершений). Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Наприклад, при поетапному процесі більшість тестів відбувається після визначення системних вимог і тоді вони реалізуються в тестових програмах. На противагу цьому, відповідно до вимог гнучкої розробки ПЗ, програмування і тестування часто відбувається одночасно.

Враховуючи аналіз проведення тестування комп’ютерних програм та усі складності його виконання, робота розробки продукту може йти по графіку або навіть прискоритись, кінцевий результат надасть змогу користувачам без проблем використовувати продукт.

## 4.2 Опис процесів тестування

Тестування виконується зазвичай на декількох етапах життєвого циклу, коли розробляється програмне забезпечення. Технологія тестування програмного забезпечення містить такі етапи:

* визначення функціоналу, що підлягає та не підлягає тестуванню;
* формування підходів, що будуть використовуватись для даного

продукту;

* написання тест кейсів;
* розробка критерію проходження тестів;
* визначення вимог середовища проведення тестування;
* проведення тестування та оцінка результатів;
* звітність результатів.

Системні вимоги:

- операційна система GNU/Linux;

- наявність JVM 17 або вище;

- наявність Angular/cli 9 або вище.

Апаратні вимоги:

Оскільки система призначена для запуску webpack development server то непотрібно визначати системні вимоги оскільки апаратне забезпечення необхідне для зручної розробки Angular проектів набагато перевищує потреби ng-deck. Проте все ще є деякі вимоги:

* наявність вільної пам'яті для встановлення програми;
* наявність вільного COM порту

Тестування проводилося на персональному комп'ютері з процесором Intel® Xeon e5-2640v3 у якого 8 ядер, 16 потоків із частотою 2.6 GHz з бустом до 3.4Ghz, 20 Мб кешу різних рівнів. На комп'ютері встановлено дві планки оперативної пам'яті DDR4 із частотою 2133MHz по 8Гб у двохканальному режимі. Відеокарта - NVIDIA GeForce GTX 1070. Операційна система - Manjaro Linux 5.15.106-1 64bit. Встановлено JDK 19 та Angular/cli 15.

## 4.3 Опис тест кейсів

### 4.3.1 unit тести

Результати тестування наведені в таблицях:

* коректність роботи Singleton класу ConfigService(таблиця 4.1)
* коректність роботи статичного фабричного методу defaultConfig класу Config(таблиця 4.2)
* коректність роботи статичного фабричного методу ofFile класу Project(таблиця 4.3)
* коректність роботи методу getArguments класу Project(таблиця 4.4)
* коректність роботи Singleton класу AngularService(таблиця 4.5)
* коректність роботи методу isActual класу FilesMonitor(таблиця 4.6)
* коректність роботи Singleton класу ProjectService(таблиця 4.7)
* коректність роботи методу formatToSend класу ProjectUtils(таблиця 4.8)
* коректність роботи методу isAngular при отриманні не директорії(таблиця 4.9)
* коректність роботи методу isAngular при отриманні валідного проекту(таблиця 4.10)
* коректність роботи методу isAngular при отриманні невалідного проекту(таблиця 4.11)

Таблиця 4.1

| Мета тесту | Перевірити коректність роботи реалізації патерну Singleton класу ConfigService |
| --- | --- |
| Опис тесту | Створення двох об'єктів за допомогою методу getInstance() та порівняння їх за адресою у пам'яті. |
| Очікуваний результат | Адреси обох об'єктів збігаються |
| Результат | Успішно |

Таблиця 4.2

| Мета тесту | Перевірити коректність створення об'єкту Config за допомогою статичного фабричного методу defaultConfig() |
| --- | --- |
| Опис тесту | Створення об'єкту Config за допомогою методу defaultConfig() та порівняння полів отриманого об'єкту з очікуваними значеннями |
| Очікуваний результат | Поля отриманого об'єкту відповідають очікуваним значенням за замовчуванням |
| Результат | Успішно |

Таблиця 4.3

| Мета тесту | Перевірити коректність створення об'єкту Project за допомогою статичного фабричного методу ofFile() |
| --- | --- |
| Опис тесту | Створення об'єкту File, створення об'єкту Project за допомогою методу ofFile, порівняння полів отриманого об'єкту Project із очікуваними. |
| Очікуваний результат | Поля створеного об'єкту Project відповідають значенням полів File на основі якого було створено об’єкт. |
| Результат | Успішно |

Таблиця 4.4

| Мета тесту | Перевірити коректність роботи методу getArguments() класу Project |
| --- | --- |
| Опис тесту | Створення об'єкту Project, виклик методу getArguments() та порівняння отриманого результату з очікуваним. |
| Очікуваний результат | Отриманий масив відповідає очікуваному результату. |
| Результат | Успішно |

Таблиця 4.5

| Мета тесту | Перевірити коректність роботи реалізації патерну Singleton класу AngularService |
| --- | --- |
| Опис тесту | Створення двох об'єктів за допомогою методу getInstance() та порівняння їх за адресою у пам'яті. |
| Очікуваний результат | Адреси обох об'єктів збігаються |
| Результат | Успішно |

Таблиця 4.6

| Мета тесту | Перевірити коректність роботи методу isActual() класу FilesMonitor |
| --- | --- |
| Опис тесту | Створення об'єкту FilesMonitor, виклик методу isActual, виклик методу isChanged() щоб дізнатися чи було змінено файли |
| Очікуваний результат | IsChanged() повертає false |
| Результат | Успішно |

Таблиця 4.7

| Мета тесту | Перевірити коректність роботи реалізації патерну Singleton класу ProjectService |
| --- | --- |
| Опис тесту | Створення двох об'єктів за допомогою методу getInstance() та порівняння їх за адресою у пам'яті. |
| Очікуваний результат | Адреси обох об'єктів збігаються |
| Результат | Успішно |

Таблиця 4.8

| Мета тесту | Перевірити коректність роботи методу formatToSend() класу ProjectUtils |
| --- | --- |
| Опис тесту | Створення об'єкту Project, передача створеного об'єкту методу formatToSend(), порівняння отриманого результату з очікуваним. |
| Очікуваний результат | Отриманий рядок відповідає очікуваному результату для відправки по послідовному порту |
| Результат | Успішно |

Таблиця 4.9

| Мета тесту | Перевірити коректність роботи методу IsAngular() класу ProjectUtils |
| --- | --- |
| Опис тесту | Створення об'єкту File, який не є директорією, передача створеного об'єкту методу IsAngular() |
| Очікуваний результат | Метод isAngular повертає false |
| Результат | Успішно |

Таблиця 4.10

| Мета тесту | Перевірити коректність роботи методу IsAngular() класу ProjectUtils |
| --- | --- |
| Опис тесту | Створення об'єкту File, який є директорією із валідним Angular проектом, передача створеного об'єкту методу IsAngular() |
| Очікуваний результат | Метод isAngular повертає true |
| Результат | Успішно |

Таблиця 4.11

| Мета тесту | Перевірити коректність роботи методу IsAngular() класу ProjectUtils |
| --- | --- |
| Опис тесту | Створення об'єкту File, який є директорією із невалідним Angular проектом, передача створеного об'єкту методу IsAngular() |
| Очікуваний результат | Метод isAngular повертає false |
| Результат | Успішно |

### 4.3.2 е2е тести

Результати тестування наведені в таблицях:

* створення конфігураційного файлу при його відсутності(таблиця 4.12)
* коректність роботи методу getDirectoriesToWatch класу ConfigService(таблиця 4.13)
* коректність роботи методу getConfig класу configService(таблиця 4.14)
* коректність роботи методу getConfig при невалідному конфігураційному файлі(таблиця 4.15)
* коректність роботи методу getProject при незаданому проекті(таблиця 4.16)
* коректність роботи методу getProject при заданому проекті(таблиця 4.17)
* коректність роботи методу getProjects класу ProjectService(таблиця 4.18)
* коректність роботи методу isListChanged при зміні файлів(таблиця 4.19)
* коректність роботи методу getProjects класу ProjectService при нестандартному конфігураційному файлі(таблиця 4.20)
* коректність роботи методу isListChanged при зміні проекту(таблиця 4.21)

Таблиця 4.12

| Мета тесту | Перевірка створення конфігураційного файлу при його відсутності у вказаному місці файлової системи |
| --- | --- |
| Опис тесту | Видалення конфігураційного файлу з файлової системи за його наявності. Створення об'єкту класу ConfigService, виклик методу getConfig() |
| Очікуваний результат | Створено конфігураційний файл із даними за замовчуванням |
| Результат | Успішно |

Таблиця 4.13

| Мета тесту | Перевірка коректності роботи методу getDirectoriesToWatch класу ConfigService |
| --- | --- |
| Опис тесту | Створення об'єкту класу ConfigService, виклик методу getDirectoriesToWatch() |
| Очікуваний результат | Отримано масив директорій які вказано у конфігураційному фалі як директорії із Angular проектами |
| Результат | Успішно |

Таблиця 4.14

| Мета тесту | Перевірка коректності роботи методу getConfig() класу ConfigService |
| --- | --- |
| Опис тесту | Створення об'єкту класу ConfigService, виклик методу getConfig() |
| Очікуваний результат | Отримано стандартний об'єкт Config |
| Результат | Успішно |

Таблиця 4.15

| Мета тесту | Перевірка коректності роботи методу getConfig() класу ConfigService при невалідному конфігураційному файлі |
| --- | --- |
| Опис тесту | Спотворення конфігураційного файлу. Створення об'єкту класу ConfigService, виклик методу getConfig() |
| Очікуваний результат | Отримано стандартний об'єкт Config, створено новий конфігураційний файл із даними за замовчуванням |
| Результат | Успішно |

Таблиця 4.16

| Мета тесту | Перевірка коректності роботи методу getProject() класу ConfigService коли проект не задано у конфігураційному файлі |
| --- | --- |
| Опис тесту | Створення об'єкту класу ConfigService, виклик методу getProject() |
| Очікуваний результат | Отримано null. |
| Результат | Успішно |

Таблиця 4.17

| Мета тесту | Перевірка коректності роботи методу getProject() класу ConfigService коли проект задано у конфігураційному файлі |
| --- | --- |
| Опис тесту | Створення нового конфігураційного файлу з вказаним проектом. Створення об'єкту класу ConfigService, виклик методу getProject() |
| Очікуваний результат | Отримано об'єкт класу Project із полями заданими у конфігураційному файлі. |
| Результат | Успішно |

Таблиця 4.18

| Мета тесту | Перевірка коректності роботи методу getProjects() класу ProjectService |
| --- | --- |
| Опис тесту | Створення об'єкту класу ProjectService, виклик методу getProjects() |
| Очікуваний результат | Отримано масив проектів що відповідають проектам які знаходяться у файловій системі за вказаним шляхом у стандартному конфігураційному файлі |
| Результат | Успішно |

Таблиця 4.19

| Мета тесту | Перевірка коректності роботи методу isListChanged() класу ProjectService |
| --- | --- |
| Опис тесту | Створення об'єкту класу ProjectService, створення пустого файлу за шляхом який відстежується, виклик методу isListChanged() |
| Очікуваний результат | Отримано false |
| Результат | Успішно |

Таблиця 4.20

| Мета тесту | Перевірка коректності роботи методу getProjects() класу ProjectService при нестандартному конфігураційному файлі |
| --- | --- |
| Опис тесту | Створення конфігураційного файлу. Створення об'єкту класу ProjectService, виклик методу getProjects() |
| Очікуваний результат | Отримано масив проектів що відповідають проектам які знаходяться у файловій системі за вказаним шляхом у конфігураційному файлі |
| Результат | Успішно |

Таблиця 4.21

| Мета тесту | Перевірка коректності роботи методу isListChanged() класу ProjectService |
| --- | --- |
| Опис тесту | Створення об'єкту класу ProjectService, створення нового проекту за шляхом який відстежується, виклик методу isListChanged() |
| Очікуваний результат | Отримано true |
| Результат | Успішно |

# Висновки

Курсовий проект присвячений розробці системи контролю веб сервера.

Тема є актуальною оскільки веб застосунки є дуже популярними а їх розробка вимагає великої кількості інструментів, одним із який є web dev сервер який призначений для запуску та відлагодження проектів. Система призначена пришвидшити запуск проектів та взяти на себе рутинні дії розробників.

У пункті «Призначення та цілі розробки системи» наведено загальні положення , опис предметного середовища та огляд існуючих технічних рішень та готових продуктів які можуть повністю виконувати задачі системи що розробляється не існує.

Крім цього сформовані всі функціональні та нефункціональні вимоги до продукту, варіанти використання, головна мета, призначення та цілі розробки системи.

У пункті «Проектування системи» поетапно описано модель роботи користувача з програмою та розроблена схема процесу роботи даного продукту. Вказано опис кожного модуля Arduino та пакета сервіса та наведені класи, з яких вони складаються. В підрозділі «Архітектура програмного забезпечення» міститься детальна інформація по функціональним модулям, з яких створена програма. Також наведений детальний опис класів, що відносяться до даного модулю, їх головні функції разом з вказаними параметрами та типом значення, що повертається. Вказані технологічні рішення та бібліотеки, їх функції, що використовуються для відтворення функціональних вимог

У розділі «Моделювання та конструювання програмного забезпечення» наведено детальний опис класів та методів розробленої системи. Наведено всі розроблені класи та методи сервіса та Arduino.

У розділі «Аналіз якості та тестування програмного забезпечення» досліджено аналіз якості програми та виявлено головні моменти, які є характерними при тестування комп’ютерної програми.

Також описано процес тестування, вказані системні та апаратні вимоги, що є необхідними для проведення даного етапу розробки продукту. Наведені вимоги до технічного та програмного забезпечення, на якому будуть проводитися тести.

Окрім цього описані детально проведені тести-кейси у вигляді таблиць.Вказано інформацію про мету тестування, принцип роботи тесту, очікуваний та отриманий результат.

Перспективою подальшої розробки є додавання підтримки інших фреймворків та систем.

# 

# Список використаних джерел

1. <https://registry.platformio.org/libraries/sstaub/LCD%20I2C>
2. <https://docs.platformio.org/en/latest/integration/ide/pioide.html>
3. <https://refactoring.guru/design-patterns/singleton>
4. <https://www.w3schools.com/cpp/cpp_pointers.asp>
5. <https://github.com/java-native/jssc/wiki/examples>
6. <https://www.javatpoint.com/log4j-example>
7. <https://projectlombok.org/features/>
8. <https://docs.arduino.cc/hardware/nano>
9. <https://www.programmingelectronics.com/serial-read/>

# Додаток А (Код сервіса)

**ConfigService.java**

package ua.lpnu.ngdeck.config;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import com.google.gson.JsonSyntaxException;

import com.google.gson.stream.JsonReader;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.models.Config;

import ua.lpnu.ngdeck.models.Project;

import java.io.\*;

import java.util.Arrays;

import java.util.Objects;

@Log4j2

public class ConfigService {

private static ConfigService configServiceSingleton;

private Config config;

private final Gson gson = new GsonBuilder().setPrettyPrinting().serializeNulls().create();

private final File configFile = new File(System.getProperty("user.home") + "/.ng-deck/config.json");

private ConfigService() {

readConfig();

}

public static ConfigService getInstance() {

if(configServiceSingleton == null) {

configServiceSingleton = new ConfigService();

}

return configServiceSingleton;

}

public String[] getDirectoriesToWatch(){

return getConfig().getDirectoriesToWatch();

}

public Project getProject(String name){

this.readConfig();

return Arrays.stream(getConfig().getProjects())

.filter(project -> Objects.equals(project.getName(), name))

.findFirst()

.orElse(null);

}

public Config getConfig() {

return config;

}

private void createConfig(){

log.info("Create default config");

try {

Writer writer = new FileWriter(configFile.getPath());

gson.toJson(Config.defaultConfig(), writer);

writer.flush();

writer.close();

} catch (IOException e) {

log.error("Error creating config", e);

throw new RuntimeException(e);

}

}

private void readConfig(){

log.info("Read config");

try {

JsonReader reader = new JsonReader(new FileReader(configFile));

config = gson.fromJson(reader, Config.class);

} catch (FileNotFoundException e) {

log.error("Config doesnt exist", e);

config = Config.defaultConfig();

createConfig();

} catch (JsonSyntaxException e) {

log.error("Invalid syntax of config file", e);

config = Config.defaultConfig();

createConfig();

}

}

}

**Codes.java**

package ua.lpnu.ngdeck.data;

public class Codes {

public static final int OK = 200;

public static final int WAS\_CHANGE = 300;

}

**Commands.java**

package ua.lpnu.ngdeck.data;

public class Commands {

public static final String CHECK\_CONNECTION = "1111";

public static final String LAUNCH\_PROJECT = "2222";

public static final String REBOOT\_PROJECT = "3333";

public static final String STOP\_PROJECT = "4444";

public static final String CHECK\_STATE = "5555";

public static final String GET\_LIST = "9999";

}

**Config.java**

package ua.lpnu.ngdeck.models;

import lombok.Builder;

import lombok.EqualsAndHashCode;

import lombok.Getter;

@Builder

@Getter

@EqualsAndHashCode

public class Config {

private String[] directoriesToWatch;

private Project[] projects;

public static Config defaultConfig(){

return Config.builder()

.directoriesToWatch(new String[]{System.getProperty("user.home")+ "/Projects"})

.projects(new Project[]{

Project.builder()

.name("")

.path("")

.runArguments("ng serve")

.build()

})

.build();

}

}

**Project.java**

package ua.lpnu.ngdeck.models;

import lombok.\*;

import java.io.File;

@Builder

@Getter

@Setter

@EqualsAndHashCode

public class Project {

private String name;

private String path;

private String runArguments;

public static Project ofFile(File file) {

return Project.builder()

.name(file.getName())

.path(file.getPath())

.runArguments("ng serve")

.build();

}

public String[] getRunArguments() {

return runArguments.split(" ");

}

}

**AngularService.java**

package ua.lpnu.ngdeck.services.Angular;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.models.Project;

@Log4j2

public class AngularService {

private static AngularService angularServiceSingleton;

private Project activeProject;

private ServerProcess serverProcess;

private AngularService() {}

public static AngularService getInstance() {

if(angularServiceSingleton == null) {

angularServiceSingleton = new AngularService();

}

return angularServiceSingleton;

}

public void start(Project project) {

log.info("start call");

this.activeProject = project;

serverProcess = new ServerProcess(activeProject);

serverProcess.start();

}

public void reboot() {

log.info("reboot call");

this.stop();

this.start(activeProject);

}

public void stop() {

log.info("stop call");

serverProcess.stopServer();

}

}

**ServerProcess.java**

package ua.lpnu.ngdeck.services.Angular;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.models.Project;

import java.io.File;

import java.io.IOException;

@Log4j2

public class ServerProcess extends Thread{

private final Project project;

private Process process;

public ServerProcess(Project project) {

this.project = project;

}

@Override

public void run() {

try {

log.info("Angular project {} start, arguments: {}", project.getName(), project.getRunArguments());

ProcessBuilder pb = new ProcessBuilder(project.getRunArguments());

pb.directory(new File(project.getPath()));

process = pb.start();

} catch (IOException e) {

log.error(e);

throw new RuntimeException(e);

}

}

public void stopServer() {

log.info("Angular project stop");

process.destroy();

}

}

**FilesMonitor.java**

package ua.lpnu.ngdeck.services.projects;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.config.ConfigService;

import java.io.IOException;

import java.nio.file.\*;

import java.util.concurrent.atomic.AtomicBoolean;

@Log4j2

public class FilesMonitor extends Thread {

private static final ConfigService configService = ConfigService.getInstance();

private AtomicBoolean isChanged = new AtomicBoolean();

public FilesMonitor() {

isChanged.set(false);

}

@Override

public void run() {

log.info("FilesMonitor Thread start");

try {

WatchService watchService = FileSystems.getDefault().newWatchService();

String[] directories = configService.getDirectoriesToWatch();

for (String path : directories) {

Path pathToDirectory = Paths.get(path);

pathToDirectory.register(

watchService,

StandardWatchEventKinds.ENTRY\_CREATE,

StandardWatchEventKinds.ENTRY\_DELETE);

}

WatchKey key;

while ((key = watchService.take()) != null) {

isChanged.set(true);

key.pollEvents().forEach(event -> log.info("Files changed {}",event.context()));

key.reset();

}

} catch (IOException | InterruptedException e) {

log.error(e);

throw new RuntimeException(e);

}

}

public boolean isChanged() {

return isChanged.get();

}

public void isActual() {

log.debug("Changes were accepted");

isChanged.set(false);

}

}

**ProjectService.java**

package ua.lpnu.ngdeck.services.projects;

import java.io.File;

import java.util.List;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.config.ConfigService;

import ua.lpnu.ngdeck.models.Project;

@Log4j2

public class ProjectService {

private static ProjectService projectServiceSingleton;

private final FilesMonitor projectsMonitor = new FilesMonitor();

private final ConfigService configService = ConfigService.getInstance();

private List<File> projects;

private boolean listChanged = false;

private ProjectService() {

projectsMonitor.start();

projects = ProjectUtils.getProjects(configService.getDirectoriesToWatch());

}

public static ProjectService getInstance() {

if(projectServiceSingleton == null) {

projectServiceSingleton = new ProjectService();

}

return projectServiceSingleton;

}

public boolean isListChanged() {

this.checkList();

return listChanged;

}

public void isActual() {

listChanged = false;

}

public void checkList() {

log.trace("checkList call");

if(projectsMonitor.isChanged()) {

List<File> newProjects = ProjectUtils.getProjects(configService.getDirectoriesToWatch());

listChanged = !newProjects.equals(projects);

log.trace("is list of projects changed: {}", listChanged);

if(listChanged) {

projects = newProjects;

}

projectsMonitor.isActual();

}

}

public List<Project> getProjects() {

this.checkList();

return projects.stream()

.map(Project::ofFile)

.map(project -> {

Project configProject = configService.getProject(project.getName());

if(configProject != null) {

if(configProject.getPath() == null || configProject.getPath().isBlank()) {

configProject.setPath(project.getPath());

}

return configProject;

}

return project;

})

.toList();

}

}

**ProjectUtils.java**

package ua.lpnu.ngdeck.services.projects;

import ua.lpnu.ngdeck.models.Project;

import java.io.File;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class ProjectUtils {

public static String formatToSend(List<Project> projects) {

List<String> projectList = projects.stream()

.map(Project::getName)

.map(str -> str.length() > 16 ? str.substring(0, 16): str)

.toList();

return projectList.toString()

.substring(1)

.replaceAll(", ", ",")

.replaceAll("]", ",");

}

public static ArrayList<File> getProjects(String[] directories) {

ArrayList<File> projects = new ArrayList<>();

for(String dir : directories) {

projects.addAll(findProjects(dir));

}

return projects;

}

private static ArrayList<File> findProjects(String path) {

File file = new File(path);

ArrayList<File> result = new ArrayList<>();

File[] subFiles = file.listFiles();

if(subFiles == null) {

return new ArrayList<>();

}

for (File subFile : subFiles) {

if (subFile.isDirectory()) {

if (isAngular(subFile)) {

result.add(subFile);

} else {

result.addAll(findProjects(subFile.getPath()));

}

}

}

return result;

}

private static boolean isAngular(File file) {

if(file.isDirectory()) {

File[] subFiles = file.listFiles();

if (subFiles != null) {

return Arrays.stream(subFiles)

.map(File::getName)

.filter(name -> name.equals("angular.json") || name.equals("node\_modules"))

.toList().size() == 2;

}

}

return false;

}

}

**ConnectionMonitor.java**

package ua.lpnu.ngdeck.services.serial;

import jssc.SerialPort;

import jssc.SerialPortException;

import jssc.SerialPortList;

import lombok.extern.log4j.Log4j2;

import java.util.Arrays;

import java.util.List;

@Log4j2

public class ConnectionMonitor extends Thread {

private String port;

private SerialPort serialPort;

private boolean isConnected = false;

@Override

public void run() {

log.info("ConnectionMonitor thread start");

while (true) {

try {

this.checkConnection();

if(!isConnected) {

this.findPort();

}

Thread.sleep(1000);

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

public SerialPort getSerialPort() {

log.trace("getSerialPort call");

try {

while (!isConnected) {

Thread.sleep(100);

}

return serialPort;

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

public String getPortName() {

return port;

}

private void findPort() {

log.trace("findPort call");

if(!isConnected) {

List<String> ports = getPorts();

for (String port : ports) {

try {

serialPort = new SerialPort(port);

serialPort.openPort();

if (serialPort.isOpened()) {

checkPort(port);

}

if (this.port != null) {

break;

}

} catch (SerialPortException | InterruptedException e) {

log.error(e);

}

}

}

}

private void checkConnection() {

log.trace("Check connection call");

boolean isOpen = false;

boolean isPortExist = false;

if(serialPort != null) {

isOpen = serialPort.isOpened();

}

if(port != null) {

isPortExist = getPorts().contains(this.port);

}

this.isConnected = isOpen && isPortExist;

log.trace("is port active: {}", this.isConnected);

}

private void checkPort(String port) throws SerialPortException, InterruptedException {

log.trace("checkPort call");

this.initPort();

Thread.sleep(2500);

String read = serialPort.readString();

log.trace("Got '{}' from {}", read, port);

if(read != null && read.startsWith("1111")) {

log.trace("port was found");

this.port = port;

}

}

private void initPort() throws SerialPortException {

serialPort.setParams(9600, 8, 1, SerialPort.PARITY\_NONE);//Set params.

}

private List<String> getPorts() {

return Arrays.stream(SerialPortList.getPortNames())

.filter(port -> port.startsWith("/dev/ttyUSB"))

.toList();

}

}

**PortReader.java**

package ua.lpnu.ngdeck.services.serial;

import jssc.SerialPort;

import jssc.SerialPortEvent;

import jssc.SerialPortEventListener;

import jssc.SerialPortException;

import lombok.extern.log4j.Log4j2;

import ua.lpnu.ngdeck.data.Codes;

import ua.lpnu.ngdeck.data.Commands;

import ua.lpnu.ngdeck.models.Project;

import ua.lpnu.ngdeck.services.Angular.AngularService;

import ua.lpnu.ngdeck.services.projects.ProjectService;

import ua.lpnu.ngdeck.services.projects.ProjectUtils;

import java.util.List;

@Log4j2

public class PortReader implements SerialPortEventListener {

private final SerialPort serialPort;

private final ProjectService projectService;

private final AngularService angularService;

public PortReader(SerialPort serialPort) {

this.serialPort = serialPort;

this.projectService = ProjectService.getInstance();

this.angularService = AngularService.getInstance();

}

@Override

public void serialEvent(SerialPortEvent event) {

if (event.isRXCHAR() && event.getEventValue() > 0) {

try {

String receivedData = serialPort.readString(event.getEventValue());

log.trace("Received {}", receivedData);

if(receivedData.startsWith(Commands.LAUNCH\_PROJECT)) {

startProject(receivedData);

}

switch (receivedData) {

case Commands.CHECK\_CONNECTION -> sendCode(Codes.OK);

case Commands.GET\_LIST -> sendList();

case Commands.REBOOT\_PROJECT -> angularService.reboot();

case Commands.STOP\_PROJECT -> angularService.stop();

case Commands.CHECK\_STATE -> checkStatus();

}

} catch (SerialPortException | InterruptedException e) {

e.printStackTrace();

}

}

}

private void checkStatus() {

if (projectService.isListChanged()) {

sendCode(Codes.WAS\_CHANGE);

} else {

sendCode(Codes.OK);

}

}

private void startProject(String receivedData) {

log.trace("startProject call");

String projectName = receivedData.substring(4);

Project project = projectService.getProjects().stream()

.filter(p -> p.getName().startsWith(projectName))

.findFirst()

.orElse(null);

angularService.start(project);

}

private void sendList() throws InterruptedException {

log.trace("sendList call");

List<Project> projectList = projectService.getProjects();

this.send(projectList.size() + ",");

this.send(ProjectUtils.formatToSend(projectList));

projectService.isActual();

}

private void sendCode(int code) {

log.trace("sendCode call");

this.send(String.valueOf(code));

}

private void send(String message) {

log.trace("Send {}", message);

try {

serialPort.writeString(message);

} catch (SerialPortException e) {

log.error("Serial Port exception", e);

throw new RuntimeException(e);

}

}

}

**SerialManager.java**

package ua.lpnu.ngdeck.services.serial;

import jssc.SerialPort;

import jssc.SerialPortException;

import lombok.extern.log4j.Log4j2;

import java.util.Objects;

@Log4j2

public class SerialManager extends Thread{

private String portName = "";

private SerialPort serialPort;

private PortReader portReader;

private final ConnectionMonitor connectionMonitor = new ConnectionMonitor();

public SerialManager() {

connectionMonitor.start();

serialPort = connectionMonitor.getSerialPort();

}

@Override

public void run() {

log.info("SerialManager Thread start");

while (true) {

if(!Objects.equals(portName, connectionMonitor.getPortName())) {

this.portName = connectionMonitor.getPortName();

serialPort = connectionMonitor.getSerialPort();

startListening();

}

}

}

private void startListening() {

log.info("Start listening on port {}" ,portName);

try {

this.portReader = new PortReader(serialPort);

serialPort.addEventListener(portReader, SerialPort.MASK\_RXCHAR);

} catch (SerialPortException e) {

log.error("Serial Exception", e);

throw new RuntimeException(e);

}

}

}

# 

# Додаток Б (Код Arduino)

**Button.h**

#pragma once

#ifndef BUTTON\_H

#include <Arduino.h>

#define BUTTON\_H

/\*\*

\* @file

\* @brief Wrapper for button

\*

\* facilitates the handling of button presses

\*/

class Button

{

private:

int pin; /\*\*<pin where the button is connected\*/

int currentValue; /\*\*<current value of button\*/

int previousValue; /\*\*<previous value of button\*/

int previousState; /\*\*<previous state of button\*/

int state; /\*\*current state of button\*/

public:

/\*\*

\* @brief Construct a new Button object

\*

\*/

Button();

/\*\*

\* @brief Initialize button

\*

\* Set pin where button is connected to read

\* @param pin pin of button

\*/

void init(int pin);

/\*\*

\* @brief Get the State object

\*

\* @return true if button is activeted

\* @return false if button is deactivated

\*/

bool getState();

/\*\*

\* @brief returns whether the state of the button has changed

\*

\* @return true if button state changed

\* @return false if button state doesn't changed

\*/

bool isChanged();

};

#endif

**ButtonProcessor.h**

#pragma once

#ifndef BUTTONS\_PROCESSOR\_H

#define buttonStopPin 6

#define buttonStartPin 7

#include <Arduino.h>

#include "Button.h"

#include "StateManager.h"

#include "Communicator.h"

#define BUTTONS\_PROCESSOR\_H

/\*\*

\* @file

\* @brief Handle buttons

\*

\*/

class ButtonsProcessor

{

private:

StateManager \*stateManager; /\*\*<StateManager reference\*/

Communicator \*communicator; /\*\*<Communicator reference\*/

List \*list; /\*\*<List reference\*/

Button startButton; /\*\*<First button\*/

Button stopButton; /\*\*<Second button\*/

/\*\*

\* @brief Handle start button

\*

\* private method to reduce amount of code in process method

\* Launch project if there is no active project or reboot

\* active project

\* @see process()

\*/

void processStartButton();

/\*\*

\* @brief Handle stop button

\*

\* private class to reduce amount of code in process method

\* Stop active project or set project which is on display as

\* active when there is no active project

\* @see process()

\*/

void processStopButton();

public:

/\*\*

\* @brief Construct a new Buttons Processor object

\*

\*/

ButtonsProcessor();

/\*\*

\* @brief Init buttons

\*

\*/

void initButtons();

/\*\*

\* @brief Handles buttons clicks

\*

\*/

void process();

};

#endif

**JoyProcessor.h**

#pragma once

#ifndef JOY\_PROCESSOR\_H

#define JoyPin A1

#include <Arduino.h>

#include "List.h"

#define JOY\_PROCESSOR\_H

/\*\*

\* @file

\* @brief Handle joystick

\*

\*/

class JoyProcessor

{

private:

List \*list; /\*\*<List reference\*/

int joy; /\*\*<current state of joystick 1 - up 0 - center -1 - down\*/

int previousJoy; /\*\*<previous state of joystick\*/

/\*\*

\* @brief read and return actual state of joy

\*

\* @return 1 if joystick was directed upwards

\* @return 0 if joystick is in the center

\* @return -1 if joystick was directed downwards

\*/

int checkState();

/\*\*

\* @brief return whether joystic state has changed

\*

\* @return true if it was changed

\* @return false if it wasnt changer

\*/

bool isChanged();

public:

/\*\*

\* @brief Construct a new Joy Processor object

\*

\*/

JoyProcessor();

/\*\*

\* @brief Handle direction of joystick

\*

\*/

void process();

};

#endif

**Communicator.h**

#pragma once

#ifndef COMMUNICATOR\_H

#include <Arduino.h>

#include "StateManager.h"

#include "List.h"

#define COMMUNICATOR\_H

/\*\*

\* @file

\* @brief communicate with PC via Serial

\*

\* Class to communicate with service on PC

\*/

class Communicator

{

private:

static Communicator \*\_communicator; /\*\*<static field to store himself for singleton\*/

StateManager \*stateManager; /\*\*<StateManager reference\*/

List \*list; /\*\*<List reference\*/

String arr[20]; /\*\*array of strings which receive from service\*/

/\*\*

\* @brief Construct a new Communicator object

\*

\*/

Communicator();

/\*\*

\* @brief process the response of service

\*

\*/

void processResponse();

public:

/\*\*

\* @brief Get the Instance object

\*

\* Singleton pattern method to get instance of class

\* @return Communicator\* single instance of class Communicator

\*/

static Communicator \*getInstance();

/\*\*

\* @brief Initialize serialPort

\*

\* @param baud speed of port

\*/

void initSerial(unsigned long baud);

/\*\*

\* @brief Establish connection with PC

\*

\*/

void establishConnection();

/\*\*

\* @brief Check if the state of Arduino is actual

\*

\* Send command to arduino to get know if the state of list or project

\* was changed from external

\*/

void checkState();

/\*\*

\* @brief Receive list from PC

\*

\* Send command to PC to get response with list of projects

\* or code that notying changed

\*/

void receiveList();

/\*\*

\* @brief launch active project

\*

\* Send command to PC to launch current active project

\* @return int code of execution

\*/

int launchProject();

/\*\*

\* @brief reboot active project

\*

\* Send command to PC to reboot current active project

\* @return int code of execution

\*/

int rebootProject();

/\*\*

\* @brief stop active project

\*

\* Send command to PC to stop current active project

\* @return int code of execution

\*/

int stopProject();

};

/\*\*

\* @brief Enumeration of commands

\*

\* Needs to unify on Arduino and Service

\*/

enum class Commands

{

CHECK\_CONNECTION = 1111, /\*\*<command to check connection with PC\*/

LAUNCH\_RPOJECT = 2222, /\*\*<command to launch project\*/

REBOOT\_PROJECT = 3333, /\*\*<command to reboot active project\*/

STOP\_PROJECT = 4444, /\*\*<command to stop active project\*/

CHECK\_STATE = 5555, /\*\*<command to check the state\*/

GET\_LIST = 9999 /\*\*<command to get list\*/

};

/\*\*

\* @brief Enumeration of codes

\*

\* Needs to unify on Arduino and Service

\*/

enum class Codes

{

OK = 200, /\*\*<Everything is OK\*/

WAS\_CHANGE = 300 /\*\*<Something changed\*/

};

#endif

**Display.h**

#pragma once

#ifndef DISPLAY\_H

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include "List.h"

#define DISPLAY\_H

/\*\*

\* @file

\* @brief Wrapper for Arduino I2C display

\*

\* Contains and control Arduino 1602 I2c display. Store

\* current state of display and change it only if it is

\* different from current to prevent it from blinking

\*/

class Display

{

private:

static Display \*\_display; /\*\*<static field to store himself for singleton\*/

LiquidCrystal\_I2C lcd; /\*\*<LiquidCrystal\_I2C object\*/

List \*list; /\*\*<reference to list\*/

StateManager \*stateManager; /\*\*<reference to stateManager\*/

String firstRow; /\*\*<first row of display which is now on the display\*/

String secondRow; /\*\*<second row of display which is now on the display\*/

/\*\*

\* @brief Construct a new Display object

\*

\*/

Display();

/\*\*

\* @brief print text on display

\*

\* Print String on display on current position

\* @param text text to print

\* @param column position where to print by horizontal

\* @param row position where to print by vertical

\*/

void print(String text, uint8\_t column, uint8\_t row);

/\*\*

\* @brief print text on first row of display

\*

\* Print text on first row of display. Compare it with

\* cuurent and print only if it is different to prevent

\* redundant blinking

\* @param text text to print on display

\*/

void printFirstRow(String text);

/\*\*

\* @brief print text on second row of display

\*

\* Print text on first second of display. Compare it with

\* cuurent and print only if it is different to prevent

\* redundant blinking

\* @param text text to print on display

\*/

void printSecondRow(String text);

public:

/\*\*

\* @brief Get the Instance object

\*

\* Singleton pattern method to get instance of class

\* @return Display\* single instance of class Display

\*/

static Display \*getInstance();

/\*\*

\* @brief Initialize the display

\*

\* Set the address of display on i2c bus and turn on

\* backlight

\*/

void init();

/\*\*

\* @brief Print logo on display

\*

\* Display hello logo on display while loading and

\* establishing connection

\*/

void showLogo();

/\*\*

\* @brief print list of projects on display

\*

\*/

void showList();

/\*\*

\* @brief print current active project

\*

\*/

void showActive();

};

#endif

**List.h**

#pragma once

#ifndef LIST\_H

#include <Arduino.h>

#include "StateManager.h"

#define LIST\_H

/\*\*

\*@file

\*@brief Contains and manipulate list of projects

\* This class contains an array of Strigns which is

\* an projects received from Service and allows to

\* get projects to output on display

\*/

class List

{

private:

static List \*\_list; /\*\*<static field to store himself for singleton\*/

StateManager \*stateManager; /\*\*<stateManager reference\*/

int size; /\*\*<amount of projects in array\*/

int offset; /\*\*<how many projects skip from start to output\*/

String list[20]; /\*\*<array of projects\*/

/\*\*

\* @brief Construct a new List object

\*

\*/

List();

/\*\*

\* @brief Get the Project object

\*

\* Return the project from list by index

\* @param index index of project in array

\* @return String project name

\*/

String getProject(int index);

public:

/\*\*

\* @brief Get the Instance object

\*

\* Singleton pattern method to get instance of class

\* @return List\* single instance of class List

\*/

static List \*getInstance();

/\*\*

\* @brief Set the Projects Array

\*

\* @param list array of strings with projects

\* @param size size of given array

\*/

void setProjects(String list[], int size);

/\*\*

\* @brief Downcrease offset

\*

\* Downcrease offset to move list up.Needed to

\* simulate movement through the menu

\*/

void moveUp();

/\*\*

\* @brief Increase offset

\*

\* Increase offset to move list down. Needed to

\* simulate movement through the menu

\*/

void moveDown();

/\*\*

\* @brief Get the First Project

\*

\* Return the first project to output on display by index offset

\* @return String project name

\*/

String getFirstProject();

/\*\*

\* @brief Get the Second Project

\*

\* Return the first project to output on display by index offset + 1

\* or String '==== END ====' if end of list

\* @return String project name

\*/

String getSecondProject();

};

#endif

**State\_Enum.h**

#pragma once

/\*\*

\* @file

\* @brief Enum of program states

\*

\*/

enum class State

{

CONNECTED, /\*\*<arduino is connected to PC\*/

DISCONNECTED, /\*\*<arduino isn't connected to PC\*/

LIST\_EMPTHY, /\*\*<list of projects is empty\*/

LIST\_CHANGED, /\*\*<list was changed and should be printed\*/

LIST\_REQUIRE\_UPDATE, /\*\*<list of projects was modified on PC\*/

LIST\_ACTUAL, /\*\*<list on display is actual\*/

PROJECT\_NONE, /\*\*<there is no projects selected\*/

PROJECT\_SELECTED, /\*\*<project selected\*/

PROJECT\_ACTIVE /\*\*project is active\*/

};

**StateManager.h**

#pragma once

#ifndef STATE\_MANAGER\_H

#include <Arduino.h>

#include "State\_Enum.h"

#define STATE\_MANAGER\_H

/\*\*

\* @file

\* @brief program state manager

\*

\* Global class to manage state of program

\*/

class StateManager

{

private:

static StateManager \*\_stateManager; /\*\*<static field to store himself for singleton\*/

State connectionState; /\*\*<state of connection to PC\*/

State listState; /\*\*<state of list\*/

State projectState; /\*\*<state of project\*/

String activeProject; /\*\*<current active project\*/

/\*\*

\* @brief Construct a new State Manager object

\*

\*/

StateManager();

public:

/\*\*

\* @brief Get the Instance object

\*

\* Singleton pattern method to get instance of class

\* @return StateManager\* single instance of class StateManager

\*/

static StateManager \*getInstance();

/\*\*

\* @brief Get the Connection State object

\*

\* @return State current state of connection

\*/

State getConnectionState();

/\*\*

\* @brief Set the Connection State object

\*

\* @param connectionState state of connection with PC

\*/

void setConnectionState(State connectionState);

/\*\*

\* @brief Get the Project State object

\*

\* @return State current state of project

\*/

State getProjectState();

/\*\*

\* @brief Set the Project State object

\*

\* @param projectState state of project

\*/

void setProjectState(State projectState);

/\*\*

\* @brief Get the List State object

\*

\* @return State current state of list on display

\*/

State getListState();

/\*\*

\* @brief Set the List State object

\*

\* @param listState state of list

\*/

void setListState(State listState);

/\*\*

\* @brief Get the Active Project object

\*

\* @return String current active project

\*/

String getActiveProject();

/\*\*

\* @brief Set the Active Project object

\*

\* @param project name of project to set as active

\*/

void setActiveProject(String project);

};

#endif

**Button.cpp**

#include "Controls/Button.h"

Button::Button() : currentValue(LOW),

previousValue(LOW),

previousState(LOW),

state(LOW) {}

void Button::init(int pin)

{

this->pin = pin;

pinMode(pin, INPUT);

}

bool Button::getState()

{

currentValue = digitalRead(pin);

if (currentValue != previousValue)

{

previousValue = currentValue;

if (currentValue == LOW)

{

state = !state;

}

}

return state;

}

bool Button::isChanged()

{

this->getState();

if (state != previousState)

{

previousState = state;

return true;

}

return state != previousState;

}

**ButtonProcessor.cpp**

#include "Controls/ButtonsProcessor.h"

ButtonsProcessor::ButtonsProcessor() : startButton(Button()),

stopButton(Button())

{

stateManager = StateManager::getInstance();

communicator = Communicator::getInstance();

list = List::getInstance();

}

void ButtonsProcessor::initButtons()

{

startButton.init(buttonStartPin);

stopButton.init(buttonStopPin);

}

void ButtonsProcessor::process()

{

if (startButton.isChanged())

{

this->processStartButton();

}

else if (stopButton.isChanged())

{

this->processStopButton();

}

}

void ButtonsProcessor::processStartButton()

{

if (stateManager->getProjectState() == State::PROJECT\_SELECTED)

{

this->communicator->launchProject();

}

else if (stateManager->getProjectState() == State::PROJECT\_ACTIVE)

{

this->communicator->rebootProject();

}

}

void ButtonsProcessor::processStopButton()

{

if (stateManager->getProjectState() == State::PROJECT\_ACTIVE)

{

this->communicator->stopProject();

this->stateManager->setListState(State::LIST\_CHANGED);

}

else if (this->stateManager->getProjectState() != State::PROJECT\_ACTIVE)

{

this->stateManager->setActiveProject(this->list->getFirstProject());

this->stateManager->setProjectState(State::PROJECT\_SELECTED);

}

}

JoyProcessor.cpp

#include "Controls/JoyProcessor.h"

int JoyProcessor::checkState()

{

int current = analogRead(A1);

if (current > 900 && joy != 1)

{

joy = 1;

}

else if (current < 100 && joy != -1)

{

joy = -1;

}

else if (current > 400 && current < 600)

{

joy = 0;

}

}

bool JoyProcessor::isChanged()

{

this->checkState();

bool changed = joy != previousJoy;

previousJoy = joy;

return changed;

}

JoyProcessor::JoyProcessor() : joy(0), previousJoy(0)

{

list = List::getInstance();

}

void JoyProcessor::process()

{

if (this->isChanged())

{

if (joy == 1)

{

this->list->moveUp();

}

else if (joy == -1)

{

this->list->moveDown();

}

}

}

**Communicator.cpp**

#include "Communicator.h"

Communicator \*Communicator::\_communicator = nullptr;

Communicator \*Communicator::getInstance()

{

if (\_communicator == nullptr)

{

\_communicator = new Communicator();

}

return \_communicator;

}

Communicator::Communicator()

{

list = List::getInstance();

stateManager = StateManager::getInstance();

}

void Communicator::initSerial(unsigned long baud)

{

Serial.begin(baud);

Serial.setTimeout(2000);

}

void Communicator::establishConnection()

{

if (stateManager->getConnectionState() == State::DISCONNECTED)

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::CHECK\_CONNECTION));

String code = Serial.readString();

if (code.length() > 0)

{

stateManager->setConnectionState(State::CONNECTED);

}

}

}

void Communicator::checkState()

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::CHECK\_STATE));

String codeStr = Serial.readString();

if (codeStr.length() > 0)

{

int code = codeStr.toInt();

if (code == static\_cast<int>(Codes::WAS\_CHANGE))

{

stateManager->setListState(State::LIST\_REQUIRE\_UPDATE);

}

}

else

{

stateManager->setConnectionState(State::DISCONNECTED);

}

}

void Communicator::receiveList()

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::GET\_LIST));

String sizeStr = Serial.readStringUntil(',');

if (sizeStr.length() > 0)

{

int size = sizeStr.toInt();

for (int i = 0; i < size; i++)

{

arr[i] = Serial.readStringUntil(',');

}

list->setProjects(arr, size);

stateManager->setListState(State::LIST\_CHANGED);

}

else

{

stateManager->setConnectionState(State::DISCONNECTED);

}

}

int Communicator::launchProject()

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::LAUNCH\_RPOJECT) + stateManager->getActiveProject());

this->stateManager->setProjectState(State::PROJECT\_ACTIVE);

}

int Communicator::rebootProject()

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::REBOOT\_PROJECT));

}

int Communicator::stopProject()

{

Serial.flush();

Serial.print(static\_cast<int>(Commands::STOP\_PROJECT));

this->stateManager->setProjectState(State::PROJECT\_SELECTED);

}

void Communicator::processResponse()

{

int response = Serial.read();

if (response == static\_cast<int>(Codes::OK))

{

Serial.println("OK");

}

else

{

stateManager->setConnectionState(State::DISCONNECTED);

}

}

**Display.cpp**

#include "Display.h"

Display \*Display::\_display = nullptr;

Display \*Display::getInstance()

{

if (\_display == nullptr)

{

Display::\_display = new Display();

}

return Display::\_display;

}

Display::Display() : lcd(LiquidCrystal\_I2C(0x27, 16, 2))

{

this->list = List::getInstance();

this->stateManager = StateManager::getInstance();

}

void Display::init()

{

lcd.init();

lcd.backlight();

this->showLogo();

}

void Display::showLogo()

{

this->printFirstRow("=== ng-deck ===");

this->printSecondRow("Connecting...");

}

void Display::showList()

{

delay(200);

this->printFirstRow(">" + this->list->getFirstProject());

this->printSecondRow(this->list->getSecondProject());

this->stateManager->setListState(State::LIST\_ACTUAL);

}

void Display::showActive()

{

this->printFirstRow("A:" + this->stateManager->getActiveProject());

this->secondRow = "";

}

void Display::print(String text, uint8\_t column = 0, uint8\_t row = 0)

{

this->lcd.setCursor(column, row);

this->lcd.print(text);

}

void Display::printFirstRow(String text)

{

if (!text.equals(firstRow))

{

this->lcd.clear();

this->print(text, 0, 0);

firstRow = text;

}

}

void Display::printSecondRow(String text)

{

if (!text.equals(secondRow))

{

this->print(text, 0, 1);

secondRow = text;

}

}

**List.cpp**

#include "List.h"

List \*List::\_list = nullptr;

List::List() : offset(0)

{

stateManager = StateManager::getInstance();

}

List \*List::getInstance()

{

if (\_list == nullptr)

{

\_list = new List();

}

return \_list;

}

String List::getProject(int index)

{

if (size > 0)

{

return list[index];

}

return "";

}

void List::setProjects(String list[], int size)

{

this->size = size;

for (int i = 0; i < size; i++)

{

this->list[i] = list[i];

}

}

void List::moveUp()

{

if (offset > 0)

{

offset = offset - 1;

stateManager->setListState(State::LIST\_CHANGED);

}

}

void List::moveDown()

{

if (offset < size - 1)

{

offset = offset + 1;

stateManager->setListState(State::LIST\_CHANGED);

}

}

String List::getFirstProject()

{

return this->getProject(offset);

}

String List::getSecondProject()

{

if (offset == size - 1)

{

return "===== END =====";

}

return this->getProject(offset + 1);

}

**StateManager.cpp**

#include "StateManager.h"

StateManager \*StateManager::\_stateManager = nullptr;

StateManager \*StateManager::getInstance()

{

if (\_stateManager == nullptr)

{

\_stateManager = new StateManager();

}

return \_stateManager;

}

StateManager::StateManager() : connectionState(State::DISCONNECTED),

listState(State::LIST\_EMPTHY),

projectState(State::PROJECT\_NONE),

activeProject("null") {}

State StateManager::getConnectionState()

{

return connectionState;

}

void StateManager::setConnectionState(State connectionState)

{

this->connectionState = connectionState;

}

State StateManager::getProjectState()

{

return projectState;

}

void StateManager::setProjectState(State projectState)

{

this->projectState = projectState;

}

State StateManager::getListState()

{

return listState;

}

void StateManager::setListState(State listState)

{

this->listState = listState;

}

String StateManager::getActiveProject()

{

return this->activeProject;

}

void StateManager::setActiveProject(String project)

{

this->activeProject = project;

}

**main.cpp**

#include <Arduino.h>

#include "Display.h"

#include "StateManager.h"

#include "Communicator.h"

#include "Controls/JoyProcessor.h"

#include "Controls/ButtonsProcessor.h"

Display \*display;

StateManager \*stateManager;

Communicator \*communicator;

JoyProcessor joyProcessor;

ButtonsProcessor buttonsProcessor;

unsigned long ButtonCnt = millis();

unsigned long ListenerCnt = millis();

void setup()

{

communicator = Communicator::getInstance();

stateManager = StateManager::getInstance();

display = Display::getInstance();

joyProcessor = JoyProcessor();

buttonsProcessor = ButtonsProcessor();

communicator->initSerial(9600);

display->init();

buttonsProcessor.initButtons();

}

void loop()

{

// If there is no connection to PC, execute untill establish

while (stateManager->getConnectionState() == State::DISCONNECTED)

{

communicator->establishConnection();

display->showLogo();

}

if (millis() - ListenerCnt > 5000)

{

communicator->checkState();

ListenerCnt = millis();

}

if (stateManager->getListState() == State::LIST\_REQUIRE\_UPDATE || stateManager->getListState() == State::LIST\_EMPTHY)

{

communicator->receiveList();

}

if (millis() - ButtonCnt > 100)

{

if (stateManager->getListState() != State::LIST\_EMPTHY)

{

if (stateManager->getProjectState() != State::PROJECT\_ACTIVE)

{

joyProcessor.process();

}

if (stateManager->getProjectState() == State::PROJECT\_ACTIVE)

{

display->showActive();

}

if (stateManager->getListState() == State::LIST\_CHANGED)

{

display->showList();

}

buttonsProcessor.process();

}

ButtonCnt = millis();

}

}