NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

HARDWARE/SOFTWARE TEST AND VALIDATION SYSTEM

PROJECT ID: **CCDS24-0193**

STUDENT NAME: **CHAN YON NAN**

PROJECT SUPERVISOR: **MR. OH HONG LYE**

EXAMINER: **DR. TAN WEN JUN**

OVERVIEW

Introduction
(Slide 3-6)

Literature Review
(Slide 7-11)

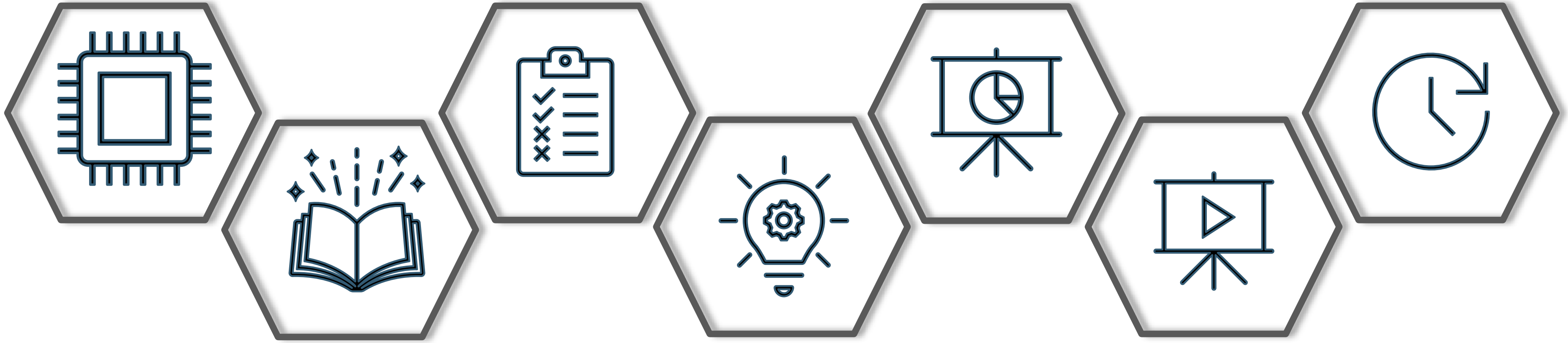
Requirement Analysis
(Slide 12-27)

Implementation
(Slide 28-47)

Result & Discussion
(Slide 47-52)

Video Demo
(Slide 53)

Conclusion & Future Work
(Slide 54-56)



INTRODUCTION

- Background
- Challenges Faced
- Project Objectives

BACKGROUND

- Multidisciplinary Design Project (MDP)
  - core module for CS & CE students
  - Develop robot car
- Duration: 10 weeks
- **STM32 microcontroller**
  - key hardware platform used in MDP projects

**STM32*

**Embedded System*

Peripherals:

Servo

Motor

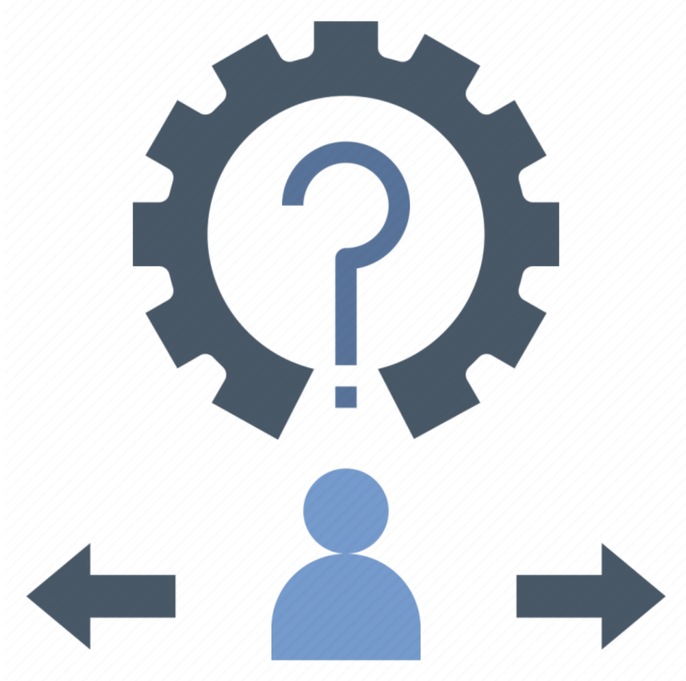
Motion Sensor

Ultrasonic sensor

IR Sensor

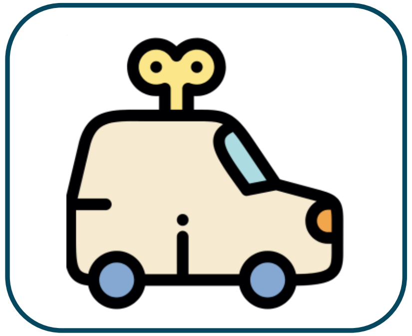
CHALLENGES FACED BY STUDENTS

- Difficulty **debugging STM32**
  - unclear software vs. hardware issue
- Time-consuming **problem isolation**
  - manual hardware checks
- Lack of **clear testing framework**
  - trial-and-error
- Frequent **back-and-forth between code and hardware**
  - frustration, delays



OBJECTIVES

1. Build MDPHelper

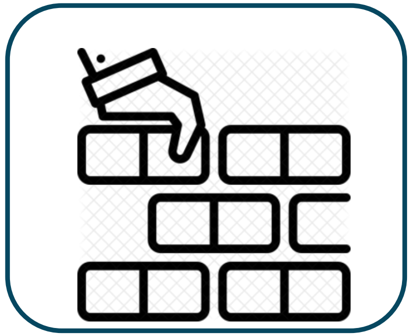


2. Provide user-friendly interface

3. Reduce manual test coding

4. Speed up debugging & testing

5. Lay groundwork for future expansion

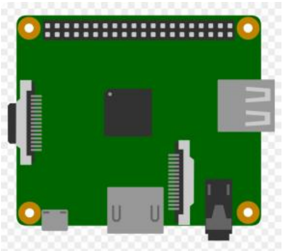


LITERATURE REVIEW

- Related Products
- Testing Approaches
- Key Design Insights

REVIEW OF RELATED TOOLS IN MARKET

- **Hardware testing kits** (e.g., Arduino, Raspberry Pi)
  - Strengths: Easy sensor/actuator testing, rapid prototyping
  - Weakness: Poor STM32 integration, extra programming needed
- **Software debugging tools** (e.g., Keil uVision)
  - Strengths: Step-by-step code debugging
  - Weakness: No real hardware validation
- **Robotic simulators** (e.g., Gazebo, Webots)
  - Strengths: Virtual testing, path planning, algorithm design
  - Weakness: No real-world hardware feedback



KEIL™
Tools by ARM

GAZEBO

Webots
robot simulation

WHAT MY PROJECT CAN LEARN

Source	Takeaway
Hardware Kits	Provide modular, easy-to-use hardware interfaces
Debuggers	Add features to help identify software bugs during hardware testing
Simulators	Include visualization tools or mock data for faster testing before hardware arrives

EMBEDDED SYSTEM TESTING APPROACHES

(Current solution)

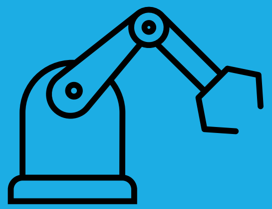
Manual testing

- Hands-on, simple, but slow and error-prone

Software simulation

- Fast, detects code issues, but misses hardware faults

(Future)



Automated validation

- Efficient, scalable, real-time fault detection, but costly + needs dev time

KEY DESIGN INSIGHTS FOR MY PROJECT

Aimed to:

User Interface

- Make it intuitive and beginner-friendly

Error Feedback

- Provide clear feedback

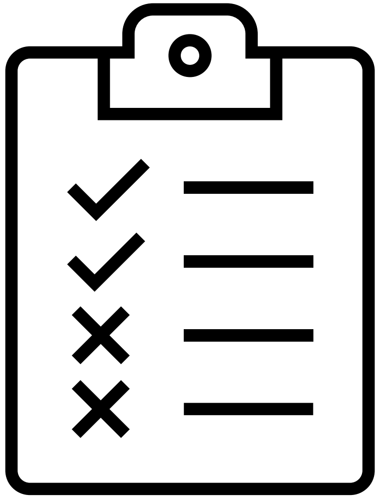
Scalability

- Plan for future automation

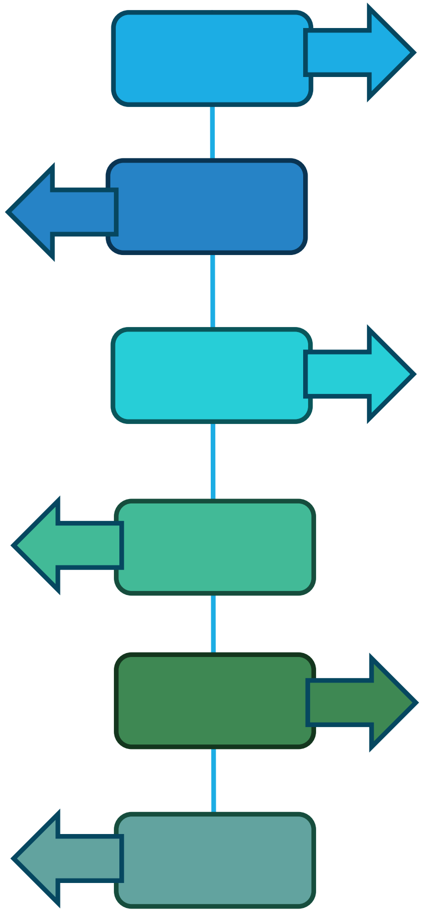
Balance

- Combine physical testing with software tools to cover both worlds

REQUIREMENT ANALYSIS & DESIGN



REQUIREMENT ANALYSIS PROCESS



Define Functionalities
Transform user requirements to functions with use case model

Functional Requirements
Define scope of work to be done

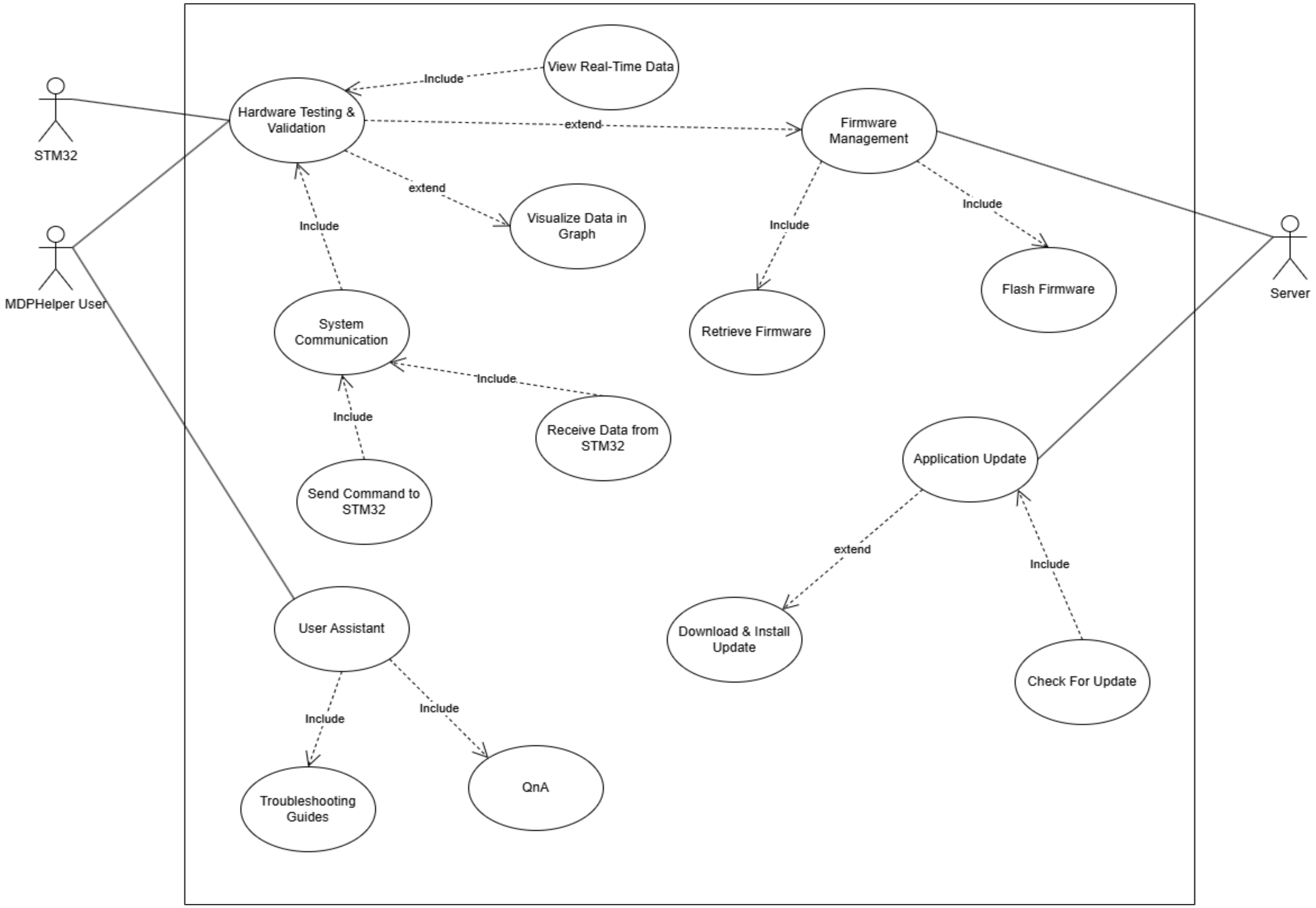
Non-Functional Requirements
Define System constraints

Technologies Consideration
Choose system frameworks, tools, and platforms

Identify Application Flow
With activity diagram

System Design

USE CASE MODEL



- Total of 15 use cases
  - Each use case represents a functionality to be implemented.

FUNCTIONAL REQUIREMENT

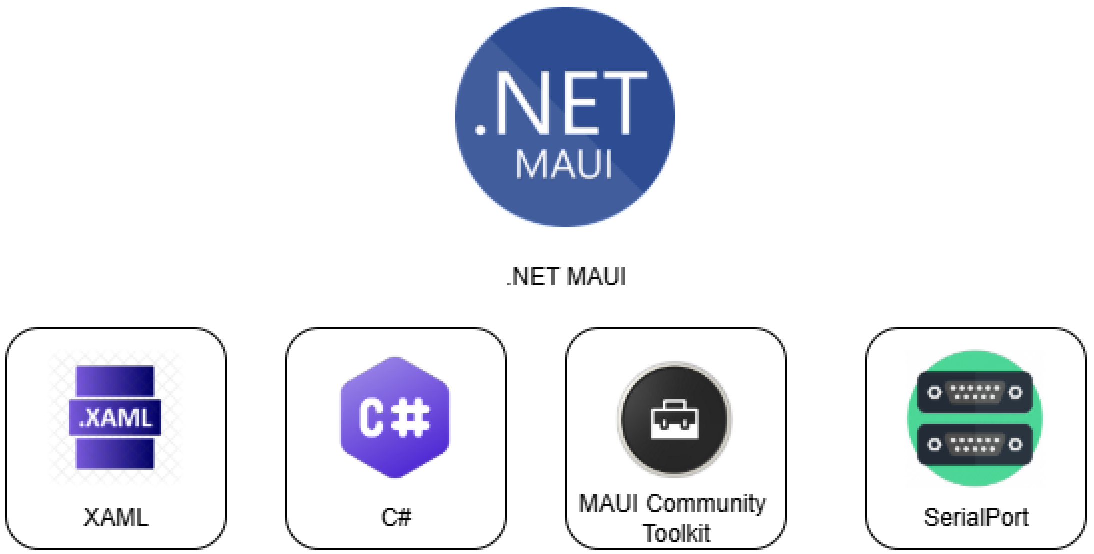
- Group the identified functionalities into the following modules.
  1. Hardware Testing & Validation module
  2. System Communication module
  3. Firmware Management module
  4. App update module
  5. User Assistance module

NON-FUNCTIONAL REQUIREMENT

1. Performance Requirements
   - Fetch firmware ≤ 2 sec
   - Flash STM32 ≤ 1 min (≤ 500 KB)
   - Display real-time logs ≤ 1 sec
   - Update graphs with no delay
2. Scalability Requirements
   - Handle 100 concurrent firmware requests
3. Usability Requirements
   - Access major functions within 2 clicks
   - Show clear, actionable error messages
   - Support common desktop/laptop resolutions
4. Reliability Requirements
   - Recover from lost connection ≤ 5 sec

TECHNOLOGIES CONSIDERATION – FRONTEND

- .NET MAUI (Desktop App)
  - Cross-platform desktop framework (Windows, macOS)
  - Single codebase, native performance
  - Supports MVU (Model-View-Update) & MVVM (Model-View-View Model) patterns
  - Mature documentation
    - ease of learning



TECHNOLOGIES CONSIDERATION – BACKEND

- .NET Core Web API
  - Lightweight, scalable REST API
  - Supports dependency injection, middleware, modular design.
- MongoDB (Database Layer)
  - Flexible NoSQL database
  - Stores firmware files, update metadata
  - Scales easily with growing data
- Docker (Containerization)
  - Simplifies deployment across environments

.NET Core

MongoDB

Docker

TECHNOLOGIES CONSIDERATION – FIRMWARE

- STM32CubeIDE
  - Coding and debugging environment
- HAL (Hardware Abstraction Layer)
  - Simplifies hardware control, improves portability
- FreeRTOS
  - Real-time task scheduling, multitasking
- ICM-20948 Library
  - 9-axis motion sensor interface (accelerometer, gyroscope, magnetometer)

STM32CudeIDE

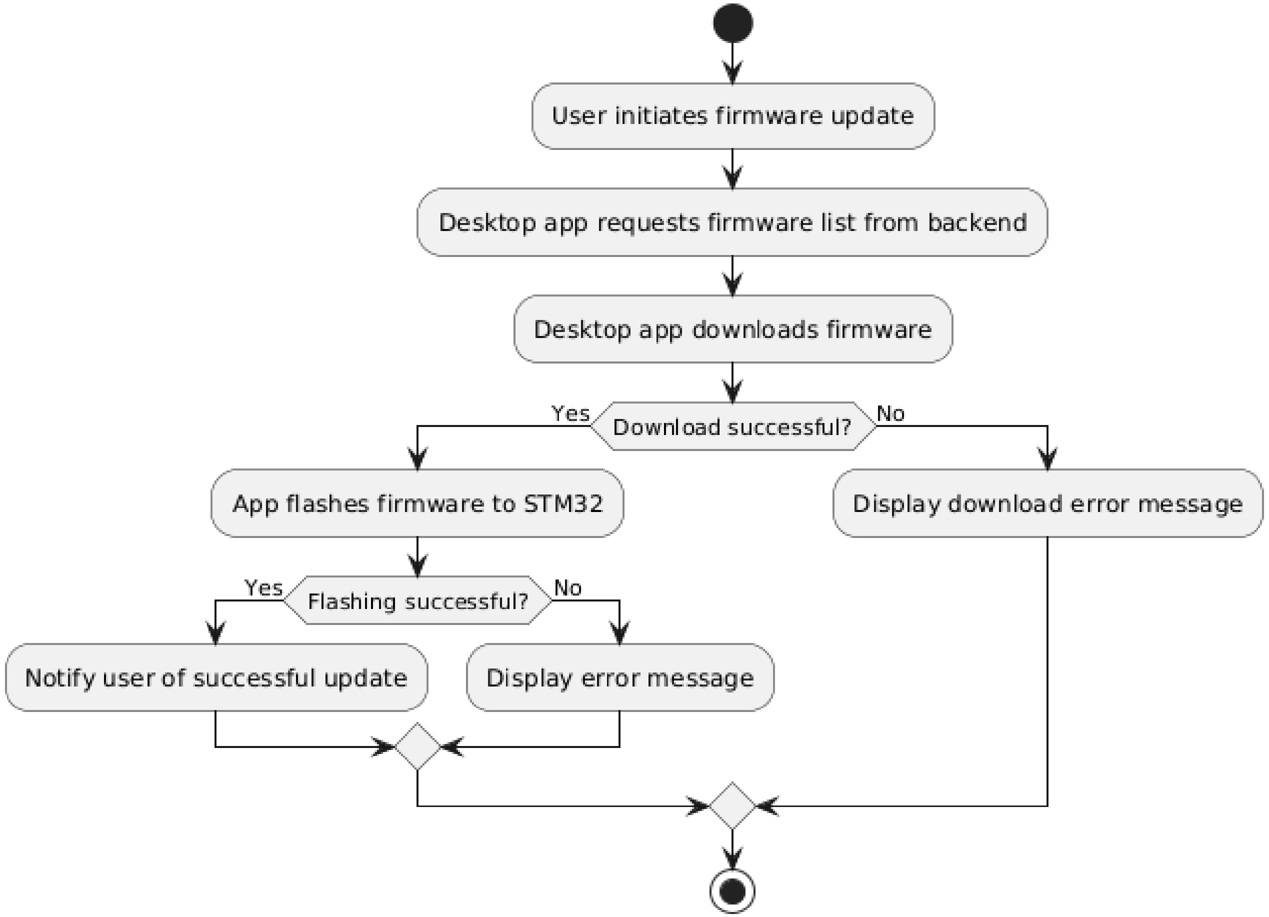
HAL

FreeRTOS

ICM-20948

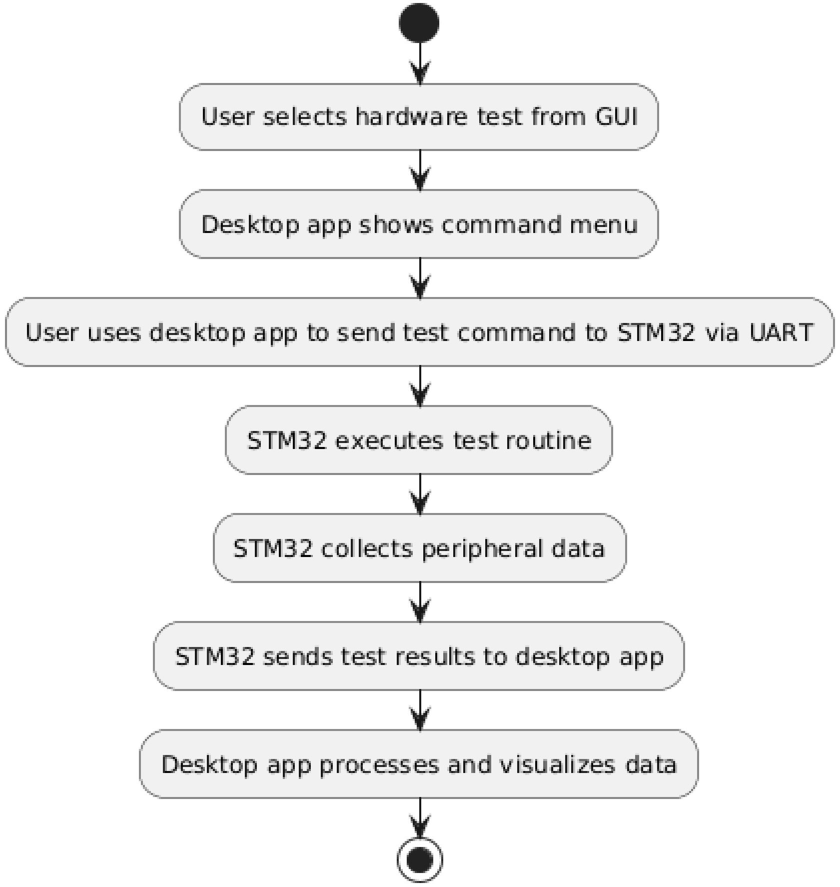
APPLICATION FLOW

- Firmware Flashing Workflow:



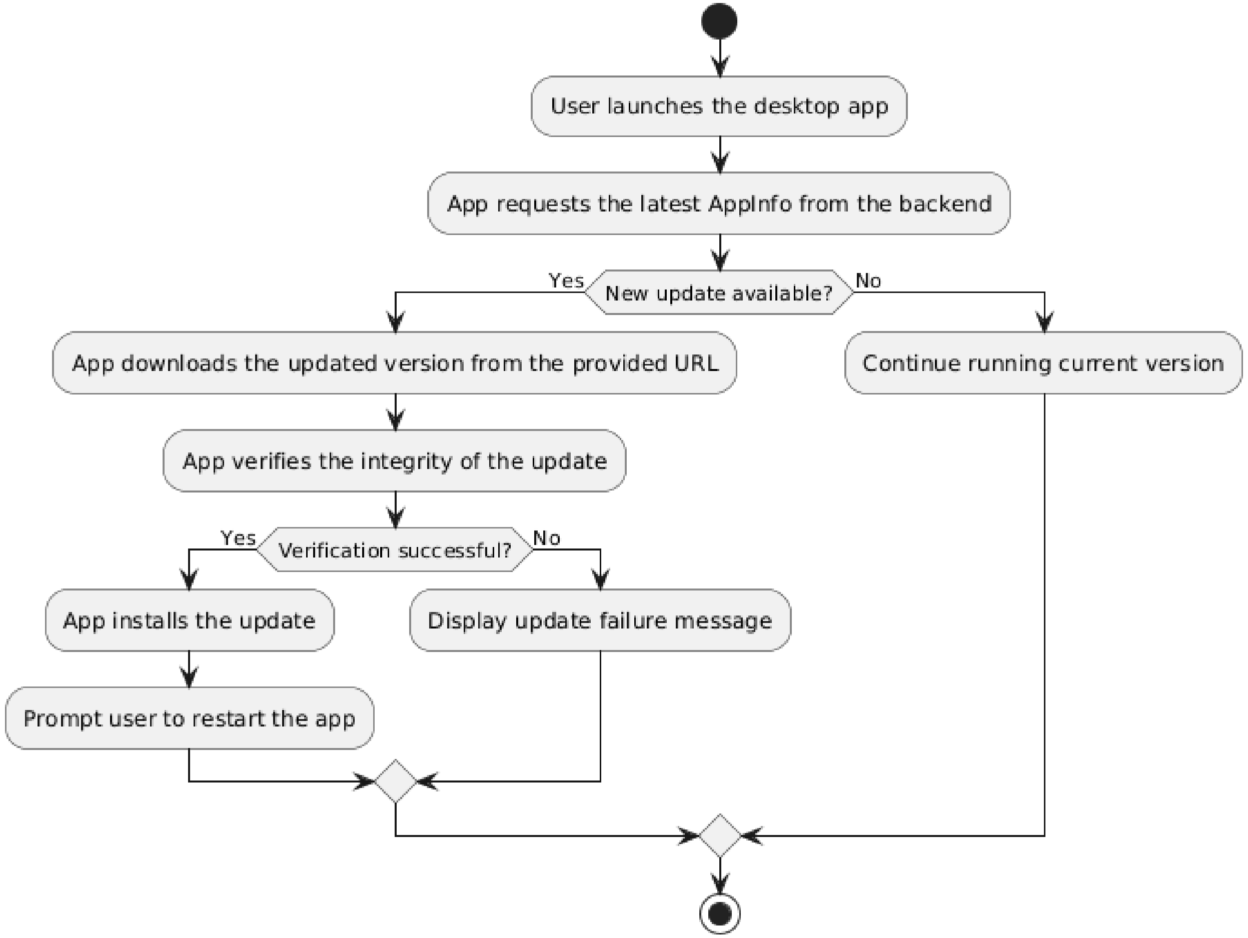
APPLICATION FLOW

- Hardware Testing Workflow:



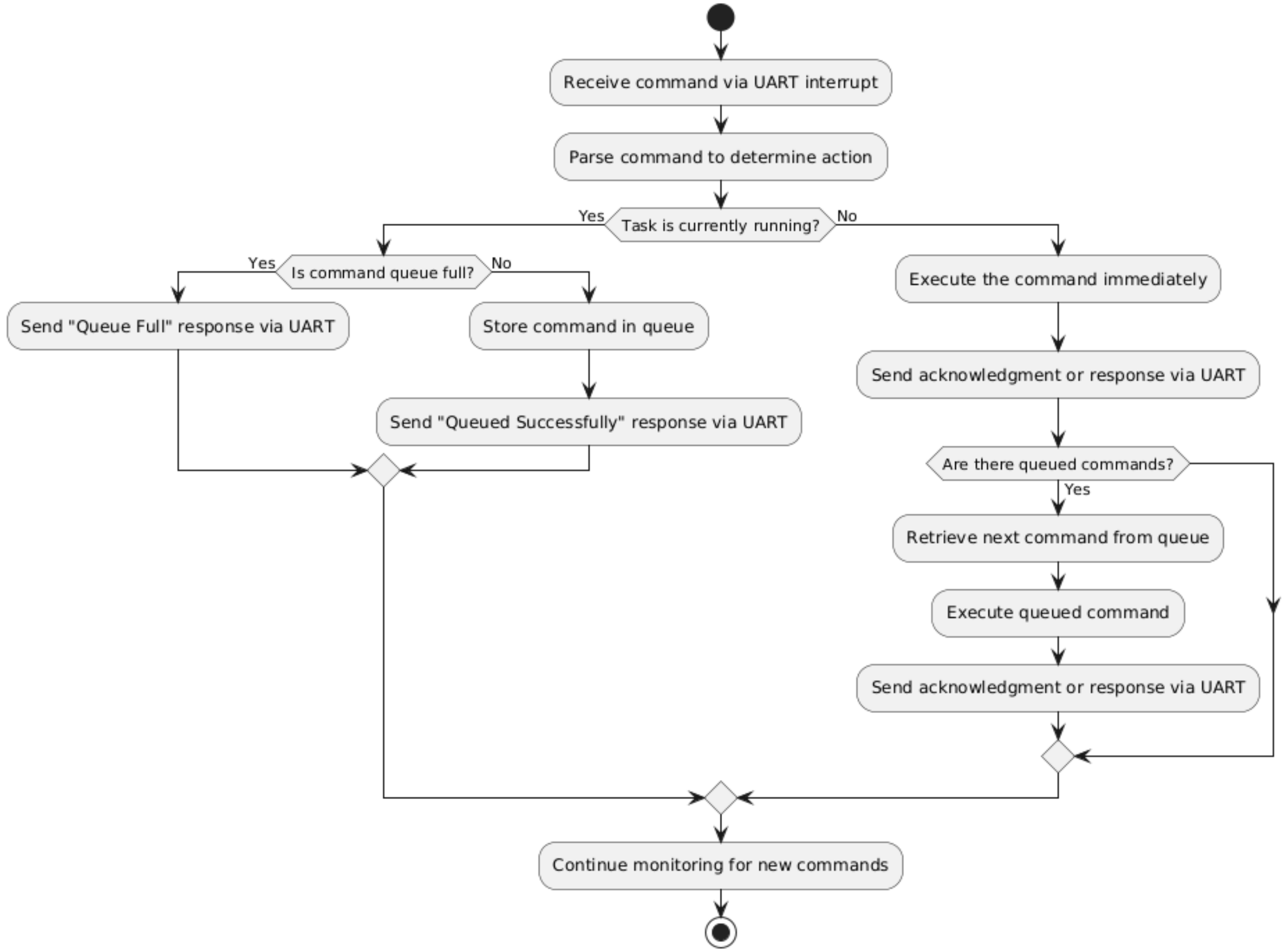
APPLICATION FLOW

- App Update Workflow:



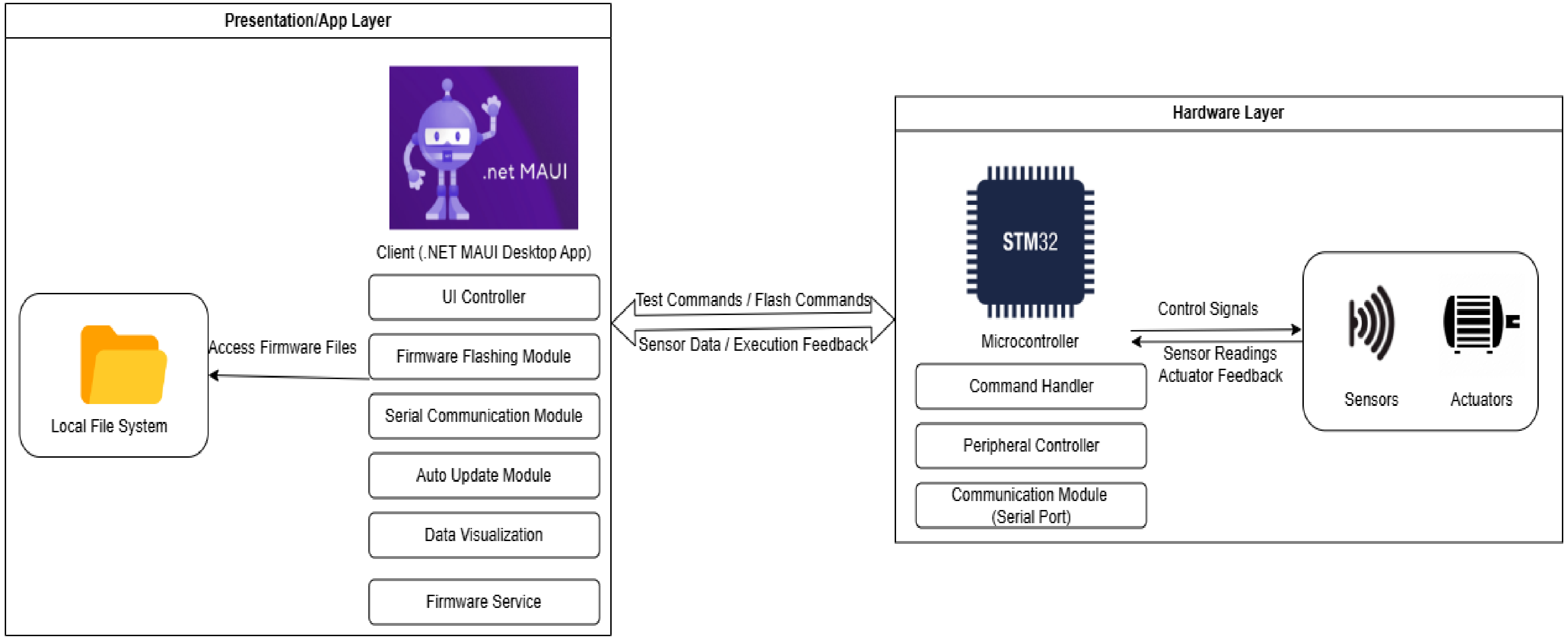
APPLICATION FLOW

- Firmware Command Processing Flow



SYSTEM ARCHITECTURE

- Local Version



High-level:

- Client–Hardware architecture

Presentation/App Layer (.NET MAUI):

- UI, firmware flashing, serial communication
- Auto updates, data visualization, firmware management

Hardware Layer (STM32):

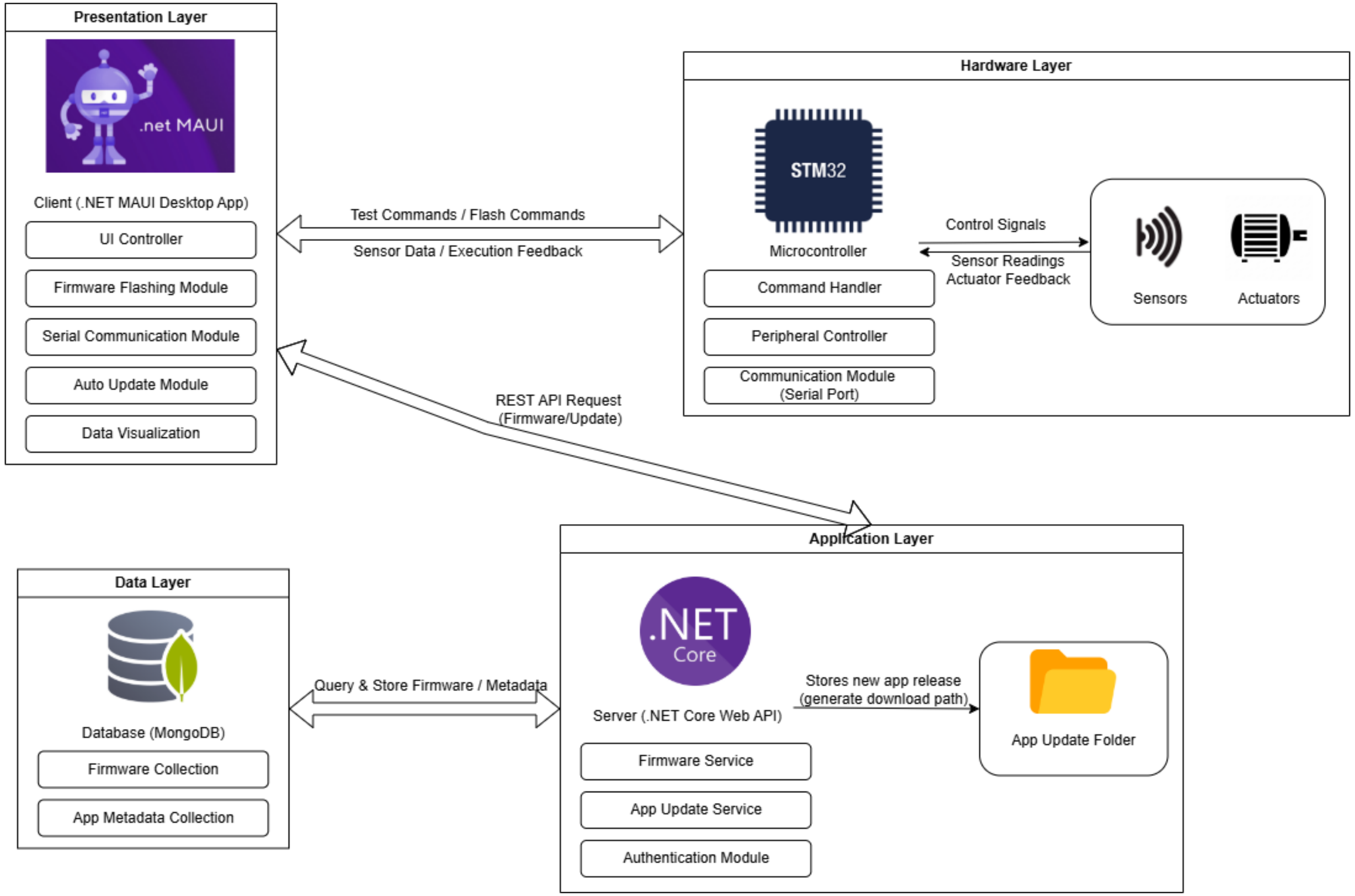
- Command handling, peripheral control, serial communication

Communication:

- App ↔ STM32: test/flash commands, sensor + actuator feedback

SYSTEM ARCHITECTURE

- Online Version



High-level:

- Multi-layer client–server–hardware architecture

Presentation Layer (.NET MAUI App):

- UI, firmware flashing, serial comm, updates, data visualization

Application Layer (.NET Core API Server):

- Firmware + update service, authentication, file storage

Data Layer (MongoDB):

- Stores firmware + app metadata

Hardware Layer (STM32):

- Handles commands, peripherals, and sensor-actuator feedback

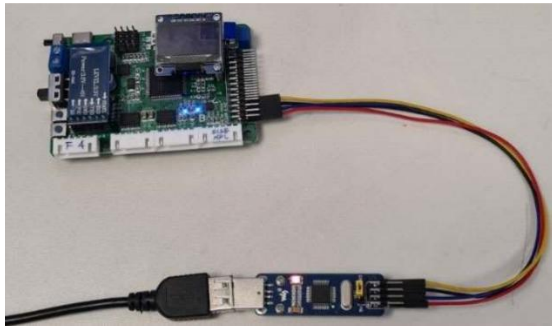
Communication:

- App ↔ STM32: test + flash commands
- App ↔ API: REST requests
- API ↔ DB: query + store firmware/metadata

STM32 AND DESKTOP APP COMMUNICATION

- UART Communication:
  - Serial command-based interaction for testing, data retrieval, flashing
- ST-LINK Programmer (USB):
  - Fast, reliable firmware flashing via USB interface
- Dual flashing options
  - more flexibility for users

**UART Connection*



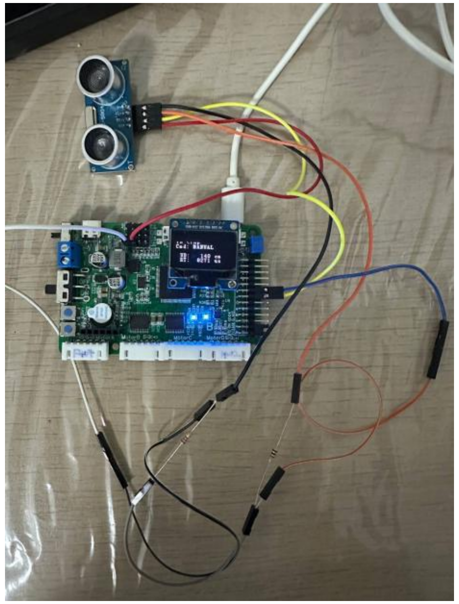
**ST-Link Connection*

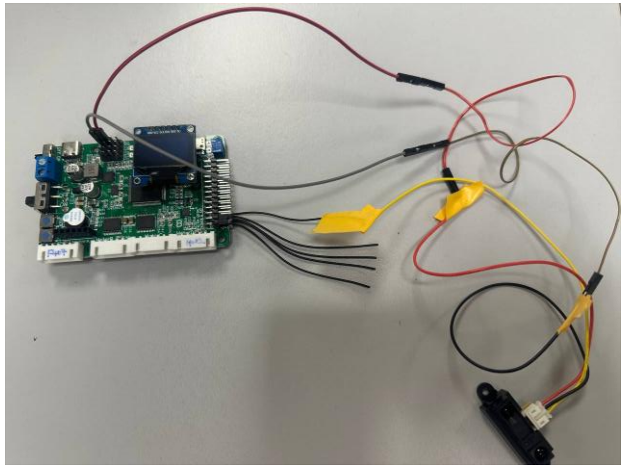
STM32 AND PERIPHERAL COMMUNICATION

- I2C (Inter-Integrated Circuit)
  - Low-speed, two-wire communication
  - Commonly used for sensors and displays (e.g motion sensor, OLED display)

- GPIO (General Purpose Input/Output)
  - Digital pins on STM32 used to read or control external devices (e.g Trigger / Echo pins on ultrasonic sensor, On/off signals to motors, servos, LEDs)

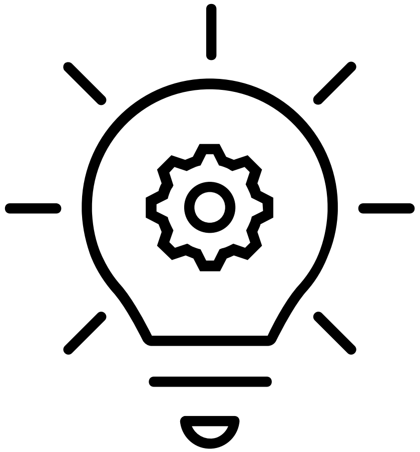
- PWM (Pulse Width Modulation)
  - A method to control analog-like behavior using digital signals (e.g Controlling motor speed, Adjusting servo position, Dimming LEDs)





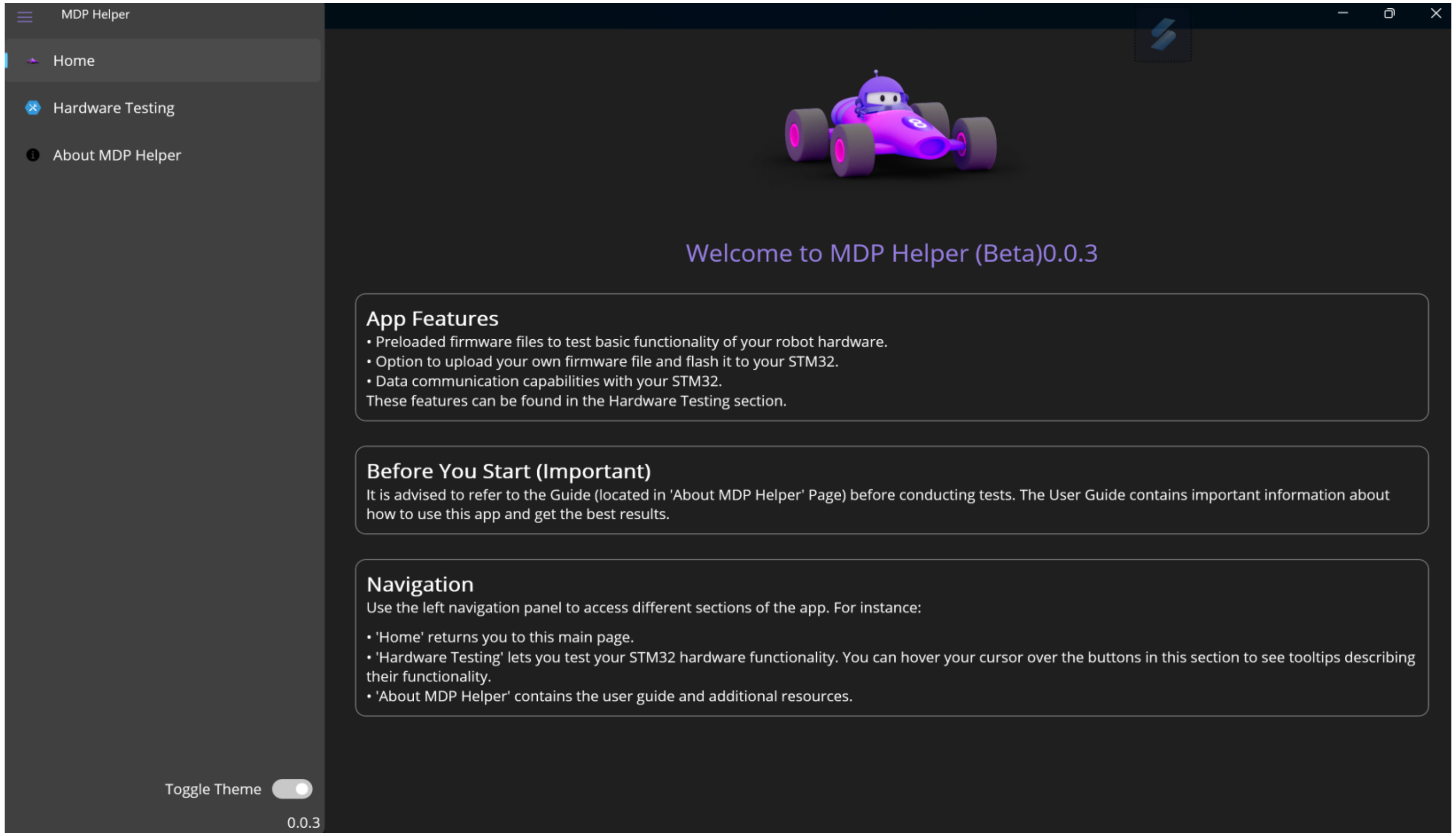
**Example connection between STM32 and Peripherals*

IMPLEMENTATION, TESTING & DEPLOYMENT



FRONTED IMPLEMENTATION

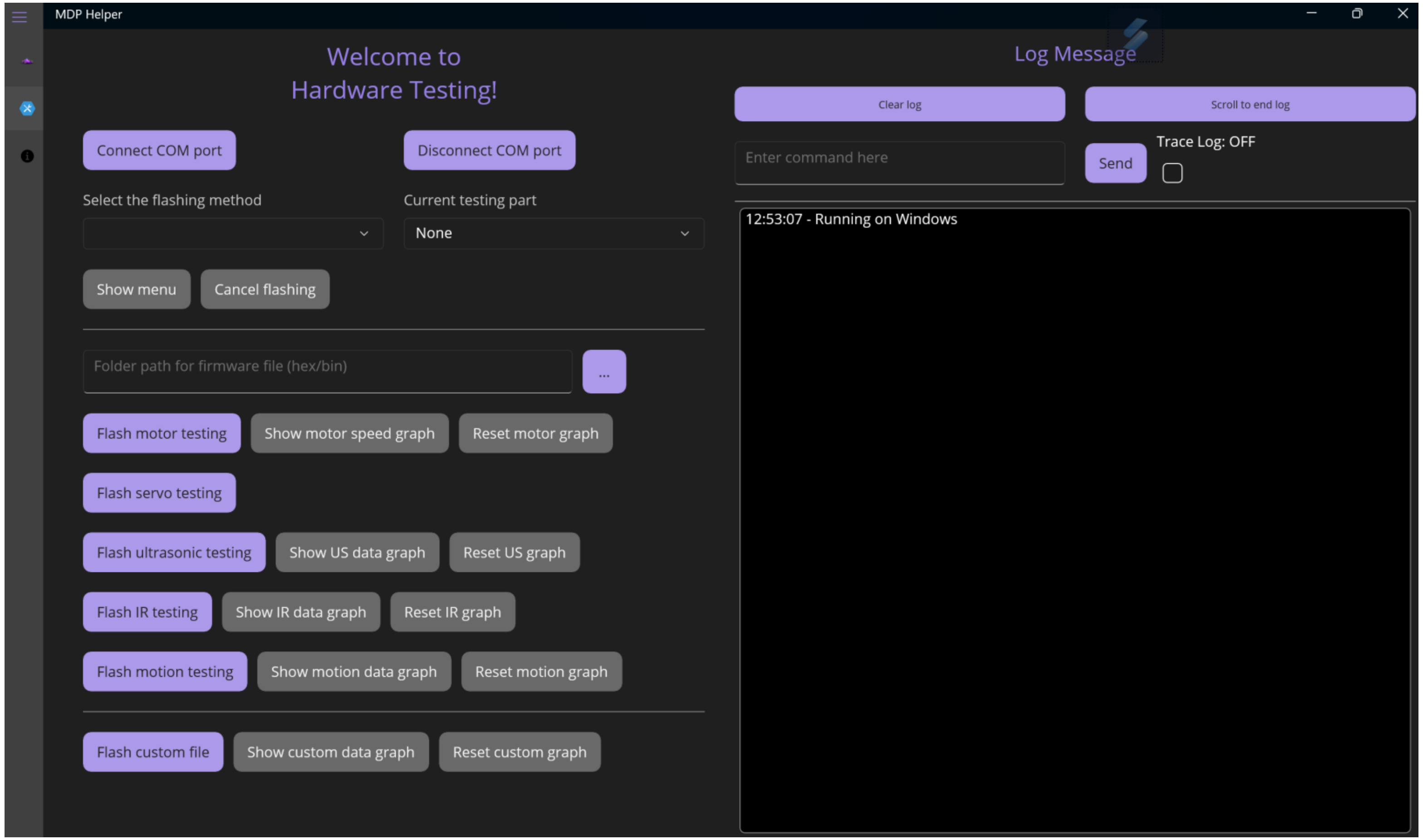
- MDPHelper – Landing Page



- Overview + navigation

FRONTED IMPLEMENTATION

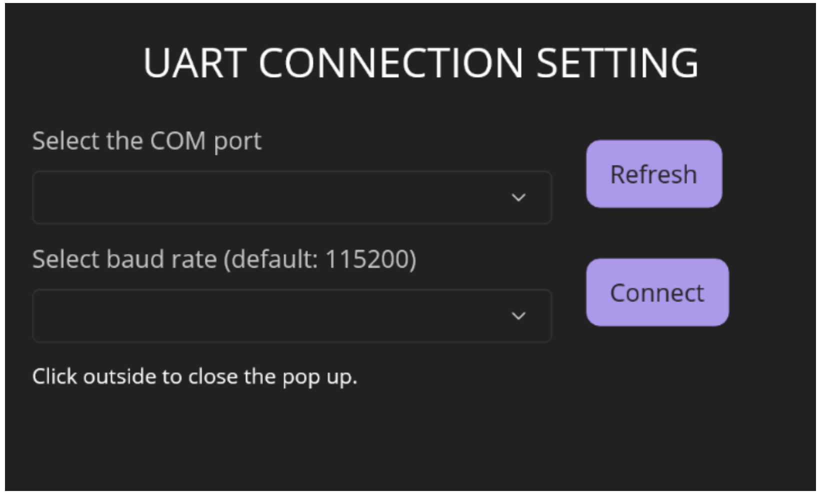
- MDPHelper – Hardware Testing Page

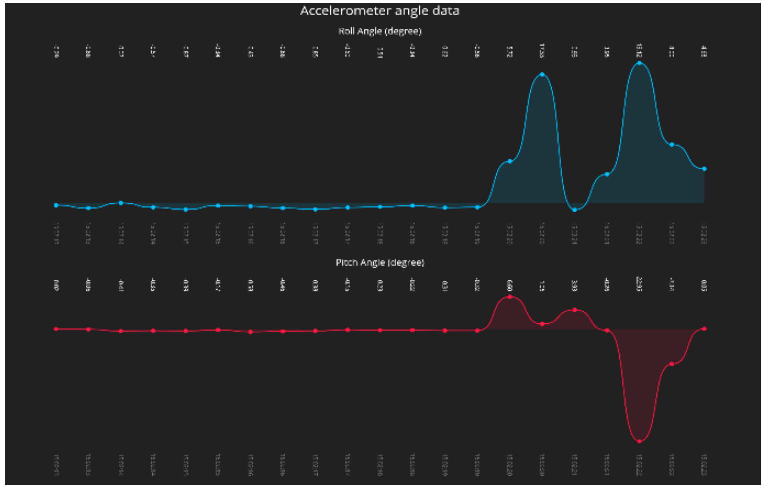


- Connect COM ports, flash firmware, monitor tests (motors, servos, sensors)

FRONTED IMPLEMENTATION

- MDPHelper – Components in Hardware Testing Page





- UART Connection Setting
- Data Visualisation

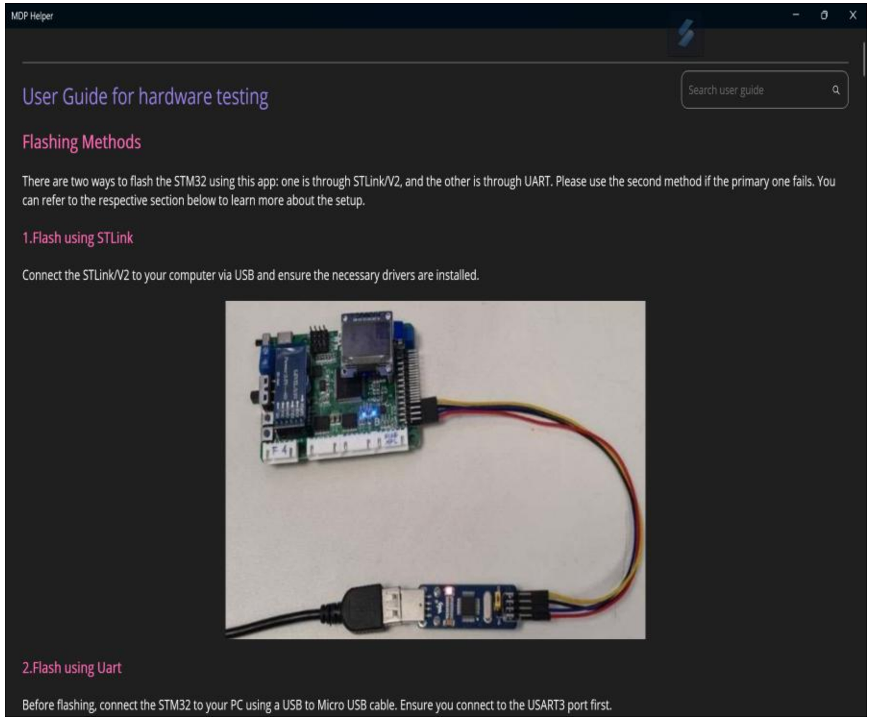
FRONTED IMPLEMENTATION

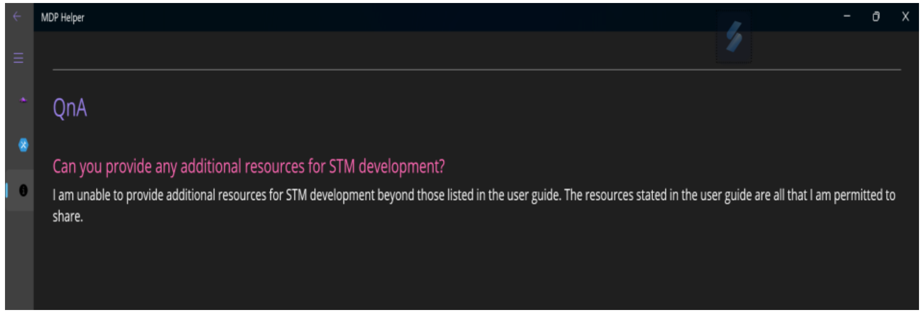
- MDPHelper – About Page

- Version info, user guide, FAQ, feedback form

FRONTED IMPLEMENTATION

- MDPHelper – Components in About Page





- User guide
- FAQ

BACKEND IMPLEMENTATION

- API Implementation (**Firmware Endpoints**)

Action	Endpoint	Description
POST	/api/firmware	Uploads new firmware
GET	/api/firmware	Retrieves all firmware
GET	/api/firmware/name/{firmwareName}	Retrieves a specific firmware by name
DELETE	/api/firmware/name/{firmwareName}	Deletes specific firmware by name
DELETE	/api/firmware/deleteAll	Deletes all firmware

BACKEND IMPLEMENTATION

- API Implementation (**AppInfo Endpoints**)

Action	Endpoint	Description
POST	/api/appinfo/upload	Uploads new app info
GET	/api/appinfo	Retrieves app information
GET	/api/appinfo/version/{version}	Retrieves specific version of the app
GET	/api/appinfo/latest	Retrieves the latest version of the app
DELETE	/filetype/{fileType}version/{version}	Deletes app info by file type and version
DELETE	/api/appinfo/deleteAll	Deletes all app info

BACKEND IMPLEMENTATION

- Backend Database Implementation

Key Fields in the Firmware Schema:

Field Name	Description
FirmwareName	The name of the firmware file.
Author	The creator or uploader of the firmware.
FileData	The binary data of the firmware file.

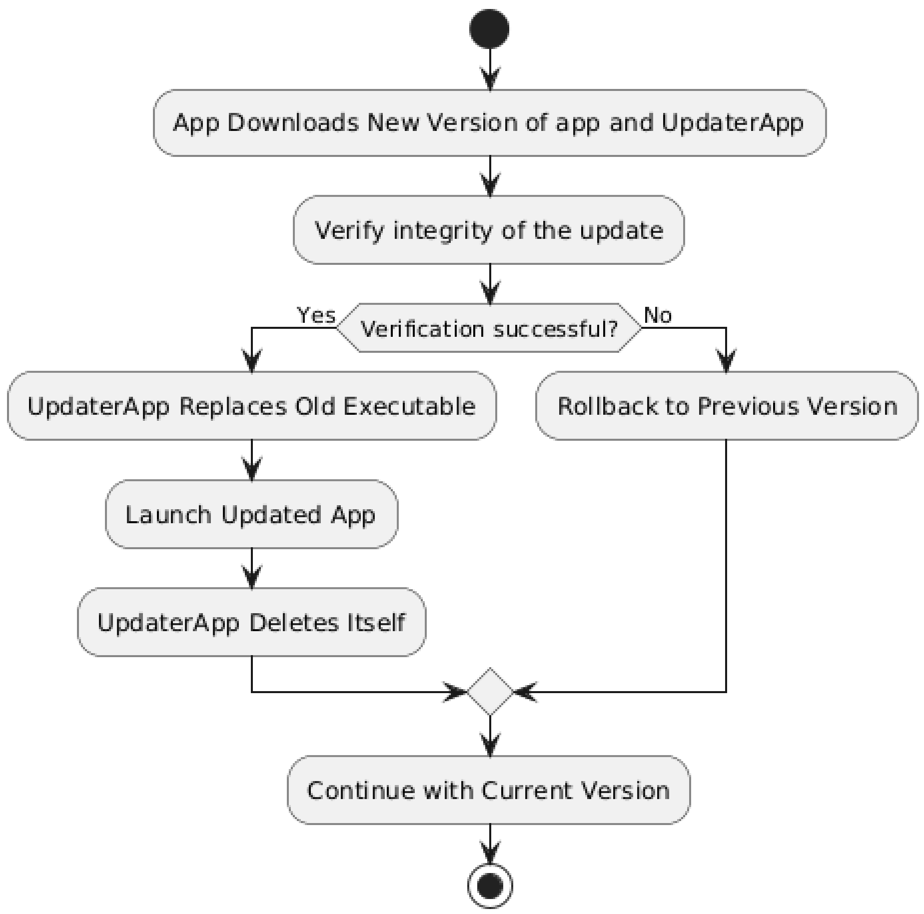
Key Fields in the AppInfo Schema:

Field Name	Description
FileName	The name of the app file.
FileType	The type of the app file (e.g., ".exe", ".pkg").
Version	The version number of the app file.
UpdatedDate	The date the app information was last updated.
DownloadUrl	The URL where the app file can be downloaded.
Author	The creator or uploader of the app.

UPDATER APP OPTIMIZATION

- Lightweight console app to manage updates in the background
- Key Benefits
  - Seamless updates
  - Minimal interaction
  - Efficiency

- **UpdaterApp Processing Flow**



FIRMWARE IMPLEMENTATION

- **Motor & Servo Control with PWM**
  - Motor Control (PWM Duty Cycle)
    - $Duty\ Cycle\ (\%) = \left(\frac{Pulse\ Width\ (CCR)}{Period\ (ARR)}\right) \times 100$
    - Higher duty cycle ⟶ faster motor speed
  - Servo Control (PWM Pulse Width)
    - $Servo\ Position\ (degrees) = \frac{Pulse\ Width\ (ms)}{20} \times 180$
    - 1 ms = 0°, 2 ms = 180°

Servo

Motor

FIRMWARE IMPLEMENTATION

- ICM-20948 Motion Sensor ⟶ 9-axis (accelerometer, gyroscope, magnetometer)
  - Accelerometer ⟶ Tilt Angles
  - Gyroscope ⟶ Orientation Change
  - Magnetometer ⟶ Heading (Yaw)

Motion Sensor

$$Row\ angle(degrees) = \arctan\left(\frac{a_y}{\sqrt{{a_x}^2 + {a_z}^2}}\right) \times \frac{180}{\pi}$$

$$Pitch\ angle(degrees) = \arctan\left(\frac{a_x}{\sqrt{{a_y}^2 + {a_z}^2}}\right) \times \frac{180}{\pi}$$

$$Tilt\ angle(degrees) = \arctan\left(\frac{\sqrt{{a_x}^2 + {a_y}^2}}{a_z}\right) \times \frac{180}{\pi}$$

$$Row\ angle(degrees) = \int g_x\, dt$$

$$Pitch\ angle(degrees) = \int g_y\, dt$$

$$Yaw\ angle(degrees) = \int g_z\, dt$$

$$Heading\ (Yaw)\ Angle\ (degrees) = \arctan\left(\frac{m_y}{m_x}\right) \times \frac{180}{\pi}$$

FIRMWARE IMPLEMENTATION

- Ultrasonic Sensor (HC-SR04) Integration
  - Measures distance to nearby objects
  - Operation
    - Trigger: 10 µs high pulse ⟶ starts ultrasonic burst (40 kHz)
    - Echo: Echo pin goes high ⟶ measures round-trip time
  - Distance Formula
    - $Distance\ (cm) = \frac{Time\ (us)}{2} \times 0.0343 cm/us$
      - Time = Echo pulse width
      - Speed of sound ≈ 0.0343 cm/µs

Ultrasonic sensor

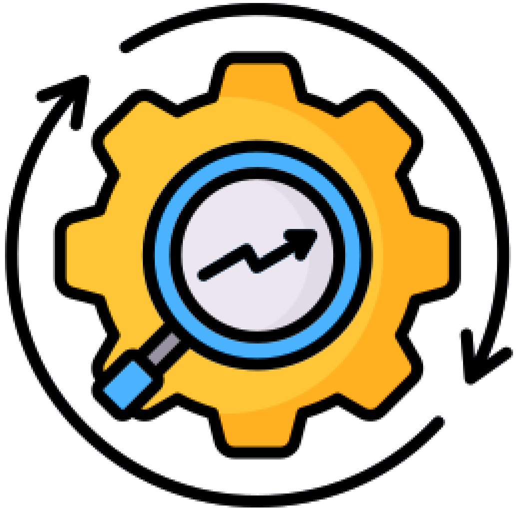
FIRMWARE IMPLEMENTATION

- Infrared (IR) Sensor Integration
  - Measures distance to nearby objects
  - Operation
    - ADC (Analog-to-Digital Converter) reads IR reflection
    - Multiple samples averaged ⟶ reduces noise
  - Distance Formula
    - $Distance\ (cm) = \frac{IR\_CONST\_A}{\frac{IR\_data\_raw\_acc}{dataPoint} - IR\_CONST\_B}$
      - IR_CONST_A, IR_CONST_B ⟶ calibration constants
      - IR_data_raw_acc ⟶ total ADC value
      - dataPoint ⟶ number of samples

IR Sensor

MEMORY AND PERFORMANCE OPTIMIZATION

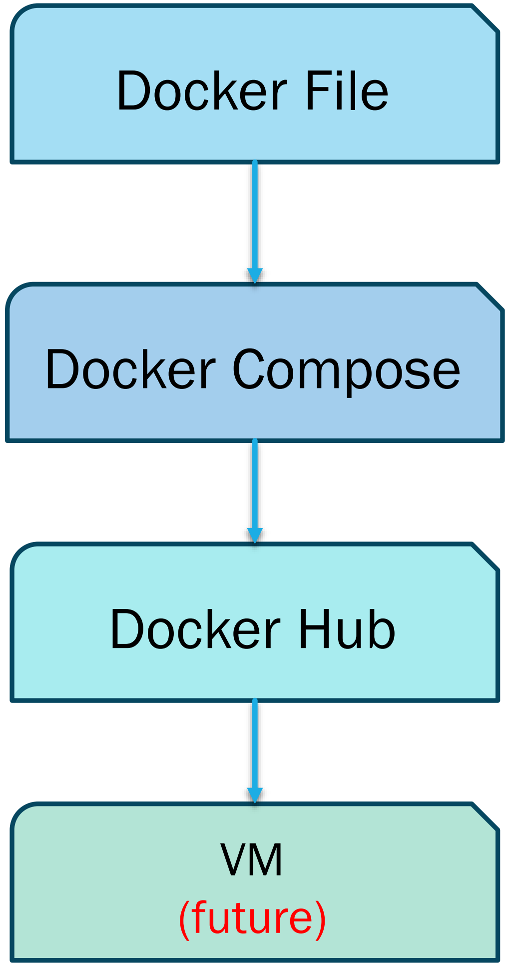
- STM32 has limited resources: 256–512 KB flash, 64–128 KB RAM
- Conditional compilation (**#ifdef**)
  - reduce code size, simplify updates
- Ring buffers
  - smooth UART data handling, avoid buffer overflows
- Optimized data parsing
  - process in chunks, reduce memory use



SYSTEM TESTING (FIRMWARE & SOFTWARE)

- Firmware Testing
  - Test motors, servos, sensors, encoder feedback on STM32
  - Check PWM speed control, sensor data, accurate distance, RPM
    - Expectation: Correct speed, angles, distance, and motor RPM
- Software Testing
  - Test MDPHelper ↔ server ↔ database
  - Verify firmware retrieval, flashing, updates, error handling
    - Expectation: Smooth firmware updates, error messages, stable flow

DEPLOYMENT (SERVER ONLY)



DEPLOYMENT (SERVER ONLY)

Docker File

- Defines how to build the server image
- Includes dependencies + environment setup
- Ensures the image is ready for deployment

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["fyp-server.csproj", "./"]
RUN dotnet restore "fyp-server.csproj"
COPY . .
WORKDIR "/src/"
RUN dotnet build "fyp-server.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "fyp-server.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "fyp-server.dll"]
```

DEPLOYMENT (SERVER ONLY)

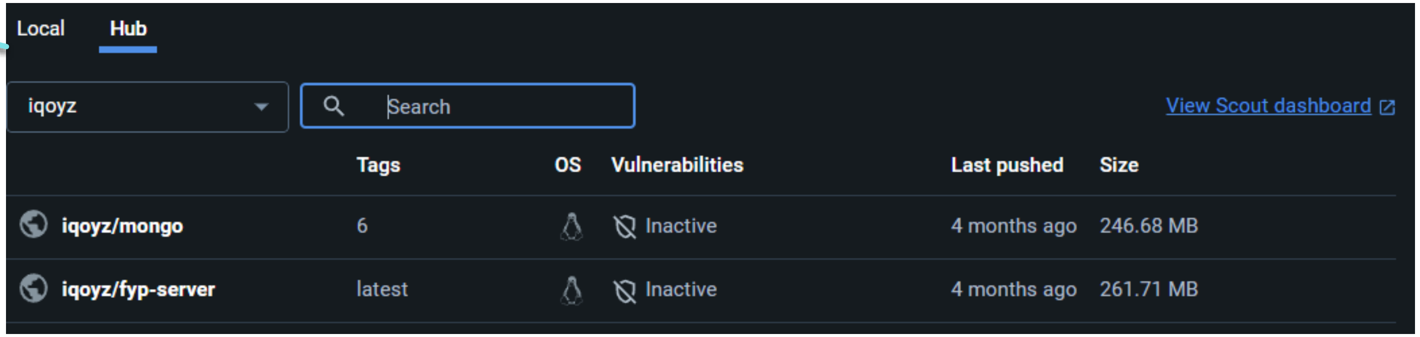
Docker Compose

- Runs **server** and **MongoDB** containers together
- Manages ports, networks, volumes
- Provides dev + prod configurations

```yaml
services:

  fyp-server:
    image: fyp-server:latest # Your server image
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080" # Expose the API on port 8080
    environment:
      - ConnectionStrings__MongoDb=mongodb://mongo:27017 # MongoDB connection string
    depends_on:
      - mongo # Ensure the MongoDB container starts first
    restart: unless-stopped
    networks:
      - fyp-network

  mongo:
    image: mongo:6 # Use the official MongoDB image
    container_name: mongo
    ports:
      - "27017:27017" # Expose MongoDB on port 27017 for external access (optional)
    volumes:
      - mongo_data:/data/db # Persist MongoDB data across container restarts
    restart: unless-stopped
    networks:
      - fyp-network

volumes:
  mongo_data: # Define a volume for MongoDB data persistence

networks:
  fyp-network:
```

DEPLOYMENT (SERVER ONLY)

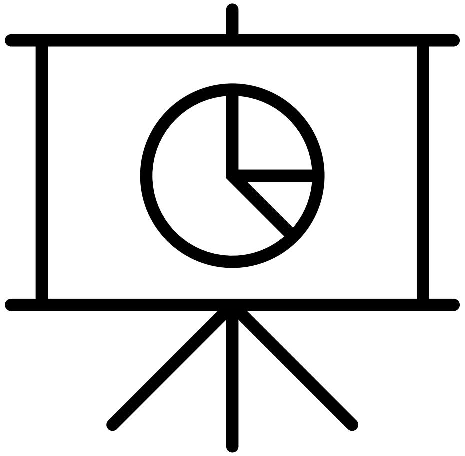
Docker Hub



- Hosts the images
- Allows you to pull + deploy the image easily on any VM
- Works like a GitHub for Docker containers

RESULTS AND DISCUSSION

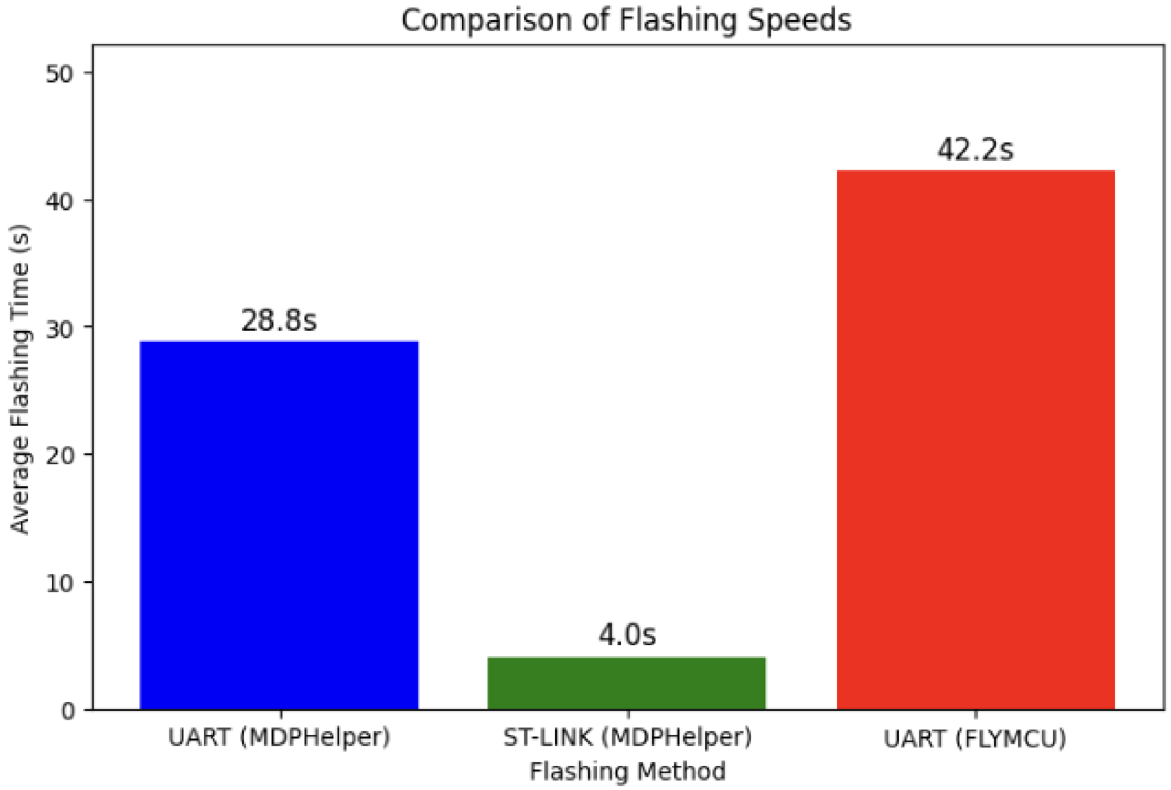
- Performance Evaluation
- User Feedback
- System Limitations



SYSTEM PERFORMANCE — FLASHING SPEED COMPARISON

- Evaluated firmware flashing across three methods:
  - MDPHelper (UART)
  - MDPHelper (ST-LINK)
  - Third-party tool: FLYMCU.exe

- MDPHelper significantly improves flashing speed compared to FLYMCU; ST-LINK delivers top performance.



SYSTEM PERFORMANCE — BACKEND AND DATABASE

Endpoint	Description	Response Time (Average)	Response Size
GET /api/firmware	Retrieves firmwares	12 ms	82.56 KB
GET /api/appinfo	Retrieves latest app update info	4 ms	150 B
Download via DownloadUrl (.exe)	Downloads Windows app update (250 MB)	~9.5 s	250 MB
Download via DownloadUrl (.pkg)	Downloads macOS app update (45 MB)	~1.8 s	45 MB

- Fast metadata retrieval and efficient file delivery.

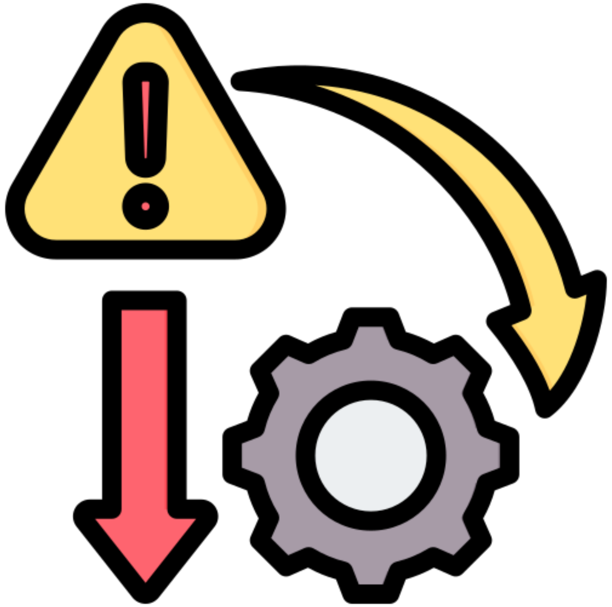
USER FEEDBACK — SUMMARY

- What Users Liked
  - Servo, IR, ultrasonic, and custom testing
  - Sensor connection guide
  - Gyro drift feature
- Issues
  - Freezing during IR tests
  - UI contrast + color readability
  - Instability with graphs

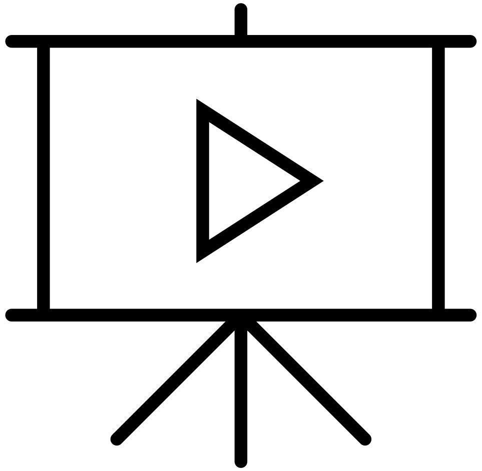
- Suggestions
  - Real-time sensor display
  - Auto-calibration
  - Shorter, clearer documentation
- Rating
  - 3.9 / 5 average
  - 40% very likely to recommend

SYSTEM LIMITATIONS

- No automated validation
  - relies on manual observation
- Functional checks only
  - lacks quantitative testing (e.g., RPM vs. PWM)
- No cross-check for encoder feedback accuracy
- No threshold alerts for abnormal sensor/actuator values
- Limited error handling + no data logging



VIDEO DEMO

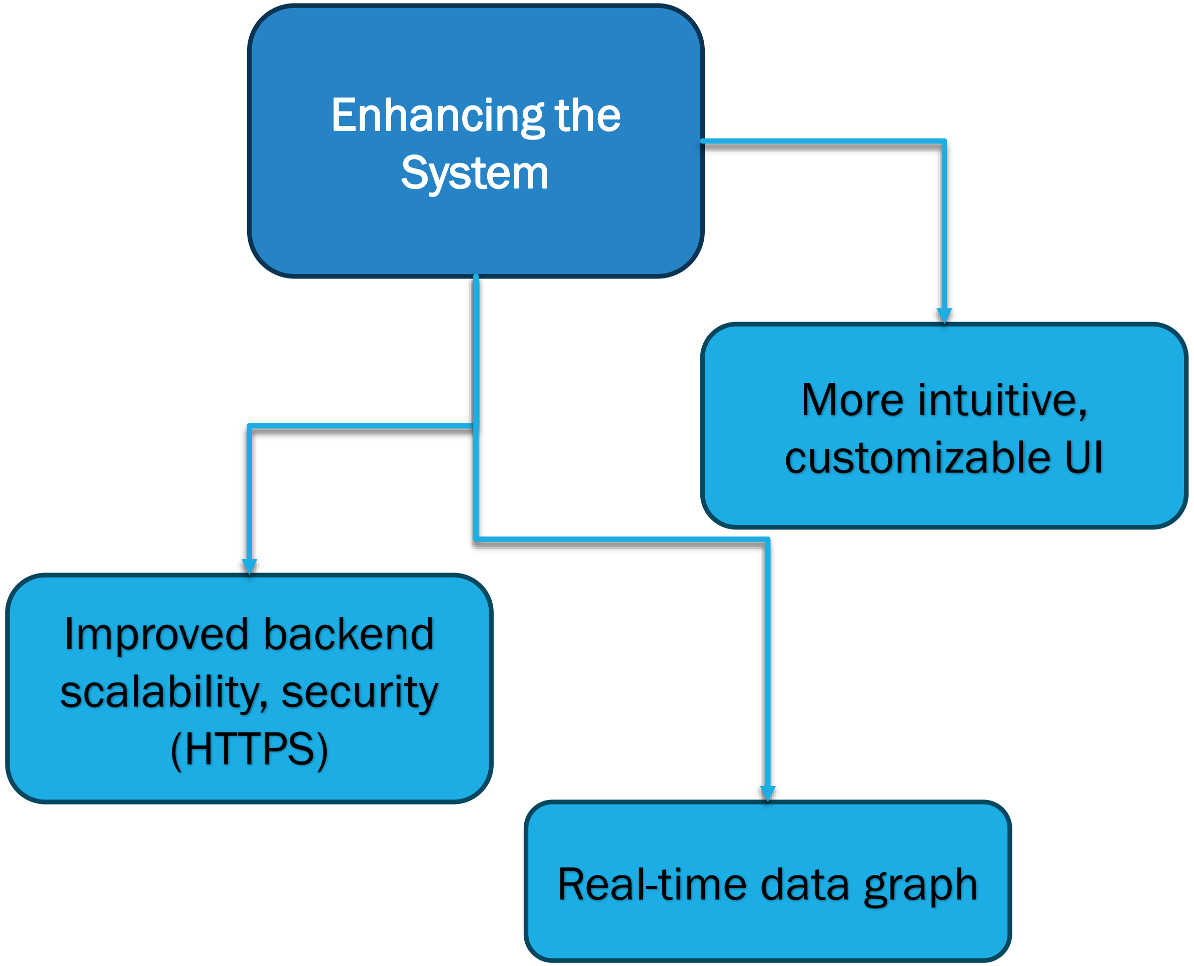


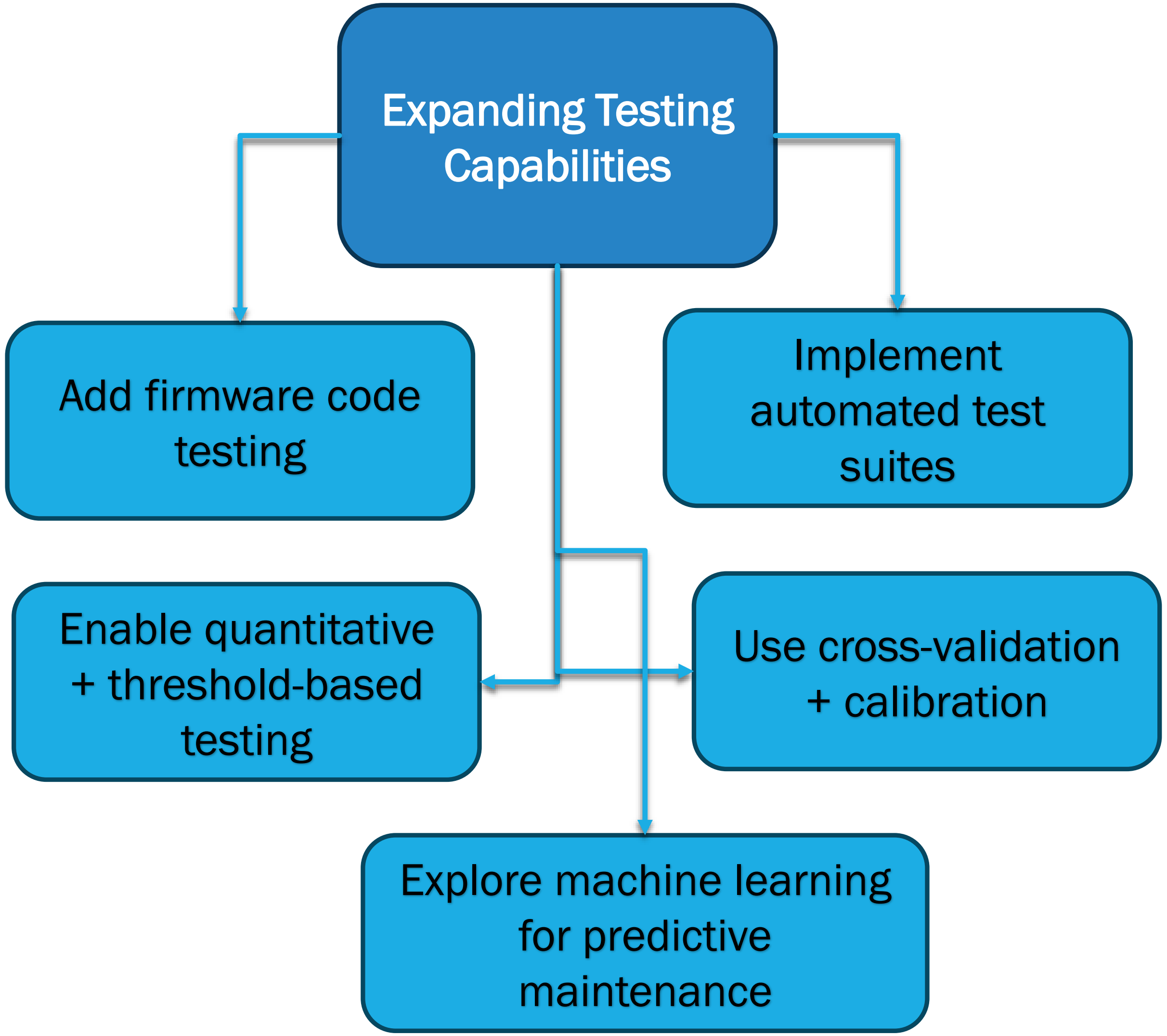
CONCLUSION & FUTURE WORK

CONCLUSION

- Developed desktop app for STM32 testing
- Streamlined testing with real-time feedback, and firmware flashing
- Positive user feedback, better performance than old tools
- Established a practical foundation for future enhancements

FUTURE WORK


Enhancing the System
More intuitive, customizable UI
Improved backend scalability, security (HTTPS)
Real-time data graph


Expanding Testing Capabilities
Add firmware code testing
Implement automated test suites
Enable quantitative + threshold-based testing
Use cross-validation + calibration
Explore machine learning for predictive maintenance

THANK YOU

Any Question?