2024

Disability Care Datalogger

FIRMWARE DOCUMENTATION

THOMAS

Table of Contents

Developer Guide

This section serves as a comprehensive guide for developers taking over or extending this project. It includes instructions on how to contribute to the codebase, modify existing functionalities, and expand the project with new features or sensors.

Project Overview

The project is built using a combination of Arduino firmware, open-source libraries, and Python-based machine learning scripts. It interfaces with MongoDB for data storage and processing, making it easy to customize and scale. The hardware includes a partially populated board with available I/O pins for additional sensors like MEMS microphones or addressable LEDs. The machine learning components are also open-source, allowing for model modifications and retraining based on new requirements.

The project can be modified using any IDE that supports .ino sketches and .cpp/.h files (e.g., Arduino IDE, VS Code with PlatformIO). Python scripts for AI processing and training are fully compatible with common data science environments such as Jupyter Notebook and PyCharm.

1. Project Structure and Key Files

    Firmware Directory (Firmware/):

        DCDLFirmwareFinalRev.ino: The primary sketch that initializes sensors, handles data acquisition, and communicates with MongoDB.
        HeartRate5.cpp and HeartRate5.h: Libraries for handling sensor inputs and pre-processing.
        Other Official open source libraries are also included and can be seen in the #Include declarations at the top of the DCDLFirmwareFinalRev.ino file.
        Note: All .cpp and .h files should follow a modular structure, making it easy to extend functionality without altering the main sketch.

    Machine Learning Directory (ML/):

        regressionmodel.py: Contains the script to train the RandomForestRegressor model. This script loads the training data (output.csv), preprocesses it, and trains the model. This model can be retrained with more info if required and will output a pkl for use with the main script.
        FinalizedModelFirmwareEnd.py: This script loads the trained model(pkl) and handles data processing and signal evaluation based on MongoDB entries.

RandomForrestModelFirmWareAutoTune.pkl: The pre-trained machine learning model saved using joblib. This file is loaded in MainProcessingModel.py to make predictions.
Note: Any modifications to the ML logic should be reflected in RandomForrestModelFirmWareAutoTune.py and the associated functions otherwise errors will arise with reading formatting etc.

Configuration Files:

Config.h: Contains configuration parameters like Wi-Fi credentials, MongoDB URI, and API keys.

2. Getting Started: Development Setup

Prerequisites:

Arduino IDE or VS Code with PlatformIO (for editing and uploading firmware).
Python 3.7+ (for executing the machine learning scripts).
MongoDB Atlas account and cluster set up (refer to the Cloud Integration Documentation for details).

Setting Up the Development Environment:

Arduino IDE:

- This is outlined in the **Installation Guide** on how to install the correct processor type and relative libraries. ○ Python Environment:
- Create a virtual environment and install the dependencies following the python documentation. Firmware Configuration:

Update the Config.h file with:

- **Wi-Fi Credentials**: SSID and PASSWORD.
- **MongoDB URI**: Connection string for the MongoDB cluster.
- **API Key**: For secure data transmission (if applicable).

3. Contributing to the Project

Understanding the Codebase:

Review the modular structure of .cpp and .h files. Each module (e.g., sensor handling, networking) is independent and can be modified without affecting other components. These are done in small modularized functions and thus as long as the main calls are still made and the functions are not changed then they will continue to work as intended.

Familiarize yourself with the DCDLFirmwareFinalRev.ino sketch for an overview of the initialization sequence and main loop.

Adding New Functionality:

Additional Sensors:

Connect new sensors to available I/O pins on the board.

Create a new .cpp and .h file for the sensor's library (e.g., new_sensor.cpp and new_sensor.h).

Initialize the sensor in the setup() function of DCDLFirmwareFinalRev.ino and add its reading logic in the loop() function.

Machine Learning Model:

Modify the training script (regressionmodel.py) with new features or datasets.

Retrain the model and save it as RandomForrestModelFirmWareAutoTune.pkl.

Load the new model in FinalizedModelFirmwareEnd.py and update the prediction logic if necessary.

Submitting Changes:

Create a new branch in the git repository with a descriptive name (e.g., feature-add-new-sensor).

Ensure the code compiles without errors and is tested on the hardware.

Open a pull request, providing detailed descriptions of the changes made.

Bug Fixes and Issue Reporting:

For each bug fix, include a description of the issue, steps to reproduce, and how it was resolved.

Use GitHub Issues or the project's issue tracking system to log bugs, feature requests, or documentation needs.

4. Extending the Project

Hardware Extensions:

The PCB has designated spots for MEMS microphones and addressable LEDs. You can populate these components and extend the functionality by updating the corresponding libraries.
Use free I/O pins for additional sensors like temperature sensors, gyroscopes, etc. Update the hardware initialization in DCDLFirmwareFinalRev.ino accordingly.

Software Extensions:

Modify the existing .cpp libraries or add new ones for complex data processing.

Expand the AI model to include new features or retrain the model to improve accuracy.

Integration with Other Services:

The project can be integrated with cloud services like AWS IoT, Google Cloud, or Microsoft Azure. Add new libraries for these services and create a new .cpp file to handle cloud communication as well as adding to the config.h for relative certificates etc. Then the DCDLFirmwareFinalRev.ino can be modified to include these changes.

5. Documentation and Knowledge Transfer

Updating Documentation:

All code changes should be accompanied by updates to the relevant documentation files.

Use the docs/ folder to create new documentation files if needed.

Machine Learning(Firmware interaction) documentation and breakdown:

ML Model Description

The machine learning model used in this project is a **RandomForestRegressor** from the sklearn library. The RandomForestRegressor is a versatile model that can handle regression tasks by building multiple decision trees and averaging their predictions to improve accuracy and control overfitting.

- **Model Objective:**
  The model is trained to evaluate the signal quality of readings from a pulse oximeter sensor. It predicts the expected heart rate based on the LED readings (e.g., green and IR LED values) collected from the sensor.

Features:

The model uses the following feature(s):

raw_values: A series of numerical values derived from the green LED sensor readings.

Expected Output:

The model outputs a continuous value representing the predicted heart rate raw values, which helps determine the quality of the signal. This output is used to guide the tuning process of the sensor parameters.

For more information on the RandomForestRegressor, refer to the sklearn RandomForest Documentation.

Training Data and Process

Training Data: The dataset used for training is a collection of sensor readings from pulse oximeters in an open source study done by Liang, Y., Chen, Z., Liu, G. *et al.* A new, short-recorded photoplethysmogram dataset for blood pressure monitoring in China. *Sci Data* **5**, 180020 (2018). https://doi.org/10.1038/sdata.2018.20 who took samples of PPG data from a number of participants and classified their risk of heart disease, this is stored in a CSV file (output.csv). Each row contains raw_values representing the green LED readings and a corresponding Heart Rate value that serves as the target variable.

Data Preprocessing:

- The raw values in the dataset are initially parsed and converted into numerical format using the parse_raw_values function.

    - Each list of raw values is exploded into individual rows to facilitate regression.
    - The data is then cleaned by dropping any rows with missing values (`NaN`), ensuring that only valid readings are used for training.
- **Splitting the Data:** The dataset is split into training and testing sets using an 80-20 split:
    - **Training Set:** 80% of the data, used to train the RandomForestRegressor.
    - **Testing Set:** 20% of the data, used to validate the model's performance.

- **Model Training:** The RandomForestRegressor is trained using the training set, with `raw_values` as the feature and `Heart Rate` as the target. After training, the model is saved as a `.pkl` file using the `joblib` library for future use.

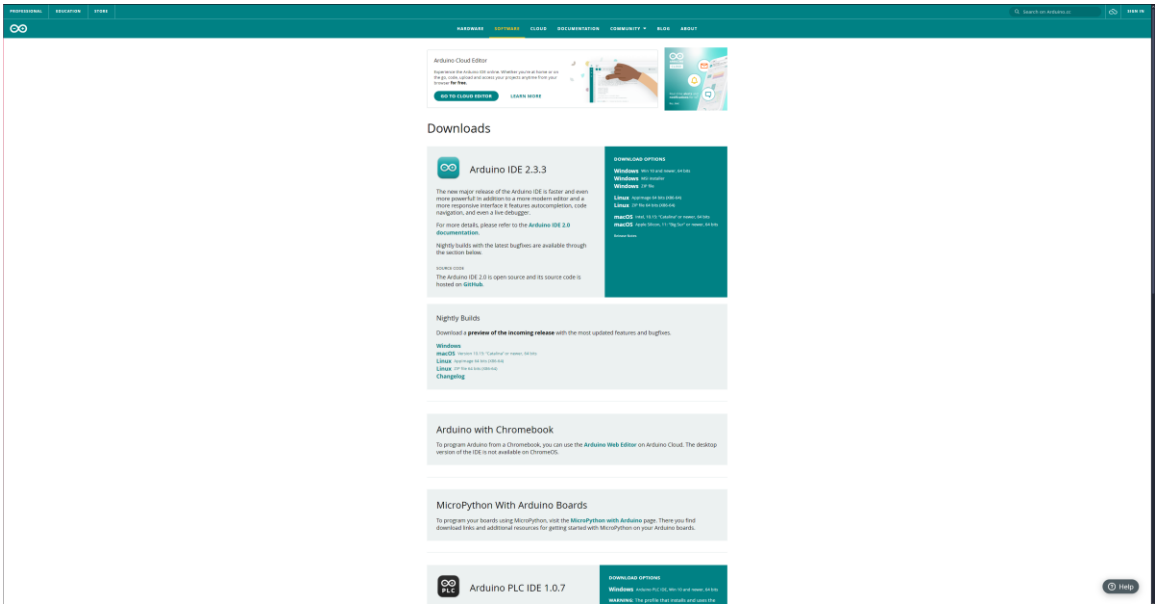
For details on the training script, refer to the code in the FinalizedModelFirmwareEnd.py and regressionmodel.py files included in the documentations folder.

Installation Guide

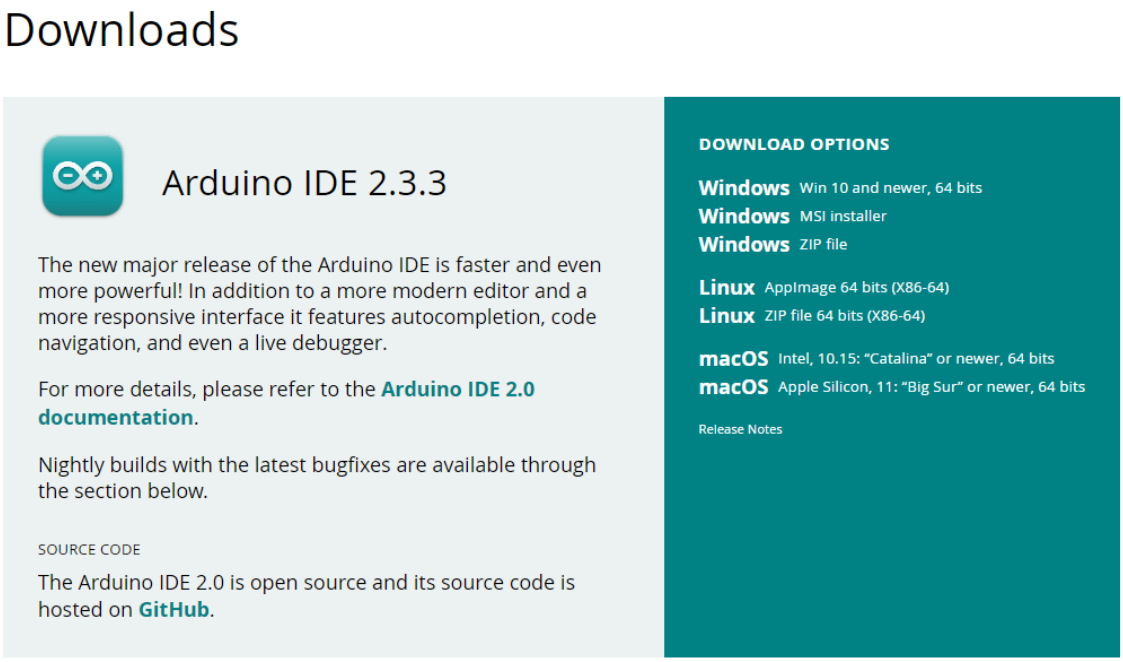
This section explains how to prepare the development environment for your project using the Arduino IDE.

1.1 Install the Arduino IDE

1. Visit the Arduino Software Page.



2. Download the appropriate version of the Arduino IDE based on your operating system (Windows, macOS, or Linux).



3. Follow the installation instructions based on your operating system:

Windows: Run the .exe file and follow the prompts to complete the installation.

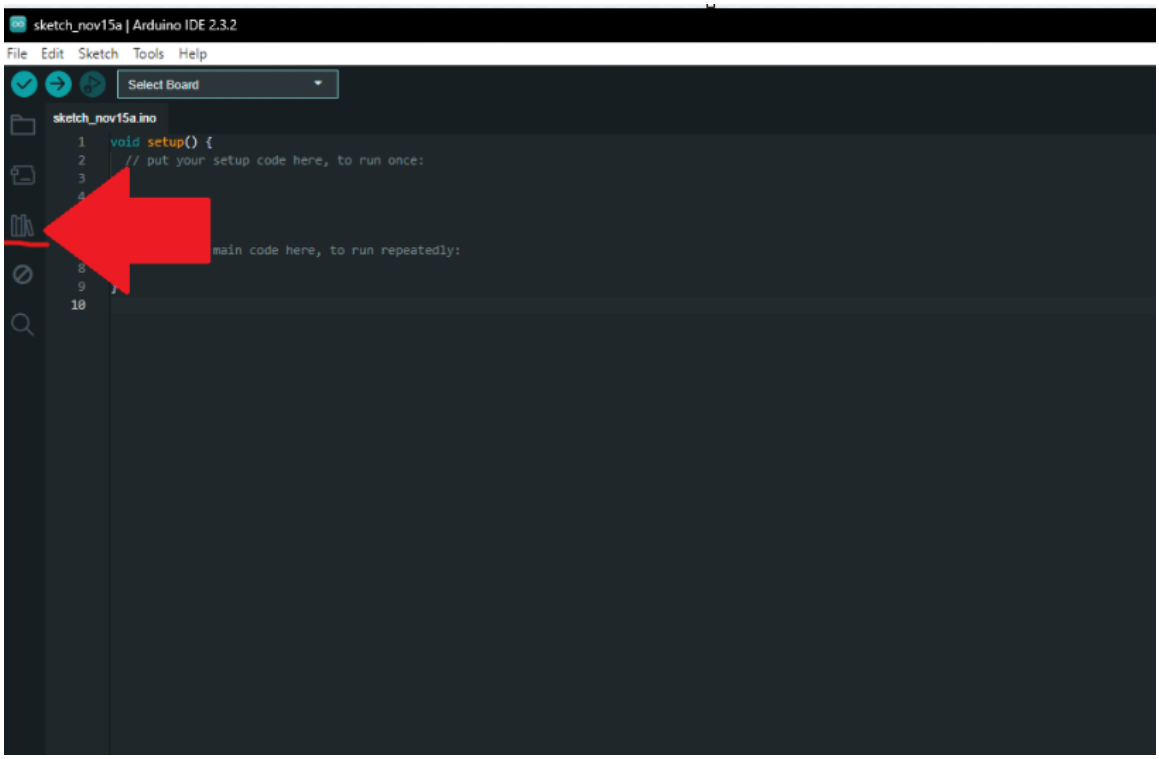
macOS: Open the .dmg file and drag the Arduino IDE to your Applications folder.

Linux: Extract the downloaded tarball and run the install.sh script.

For more detailed instructions, refer to the official Arduino Installation Guide.

1.2 Install Required Libraries

1. Open the Arduino IDE.
2. Navigate to Sketch → Include Library → Manage Libraries…



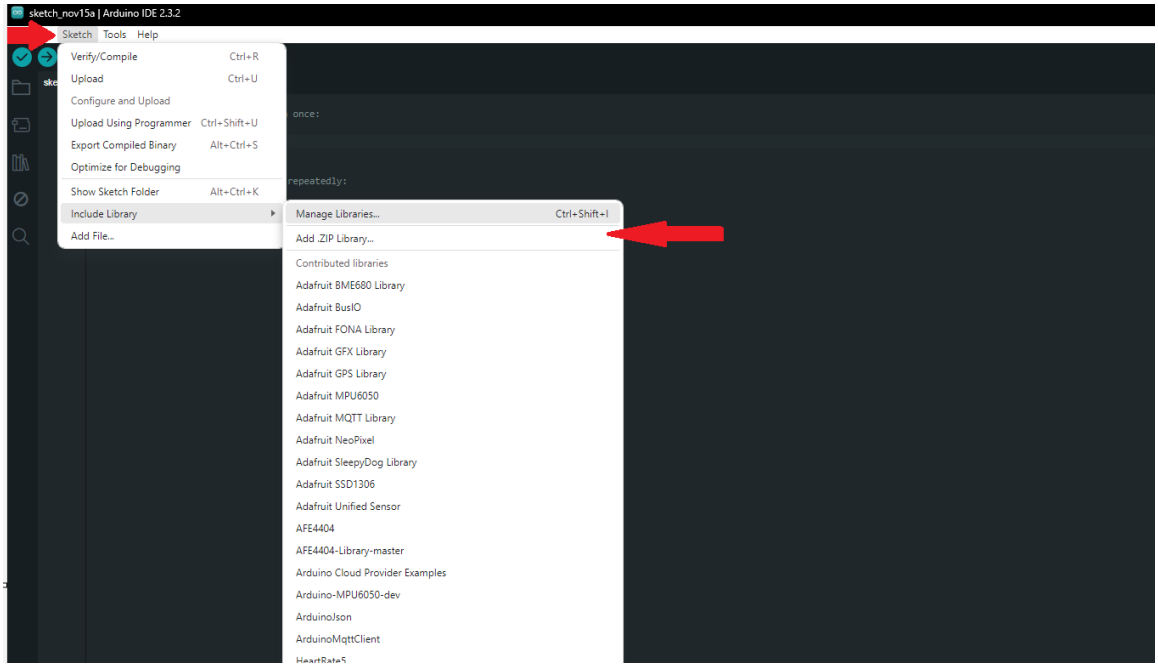
3. In the Library Manager, search for the required libraries and click "Install".
   Here's a list of the necessary libraries for this project:

```
<Wire.h>
<WiFi.h>
<HTTPClient.h>
<ArduinoJson.h>
"HeartRate5.h"
<Adafruit_MPU6050.h>
<Adafruit_Sensor.h>
<MCP79412RTC.h>
<Adafruit_NeoPixel.h>
<TimeLib.h>
```

HeartRate5 is a custom library and will need to be installed using the method below.

For custom or third-party libraries:

- Download the library as a .zip file from its source repository.
- Navigate to Sketch → Include Library → Add .ZIP Library….



- Select the .zip file and click "Open" to add it to your Arduino library.

- For Ease of use there is an included Zip file with all currently used Libraries for the firmware.

For additional details on adding libraries, refer to Adding Libraries to the Arduino IDE.

1.3 Manually Adding Libraries

Download the .zip library files from the respective repositories.

Locate your Arduino Libraries folder:

- **Windows**: Documents/Arduino/libraries
- **macOS**: ~/Documents/Arduino/libraries
- **Linux**: ~/Arduino/libraries

Extract the contents of the .zip file and copy the library folder (e.g., ArduinoJson) into the libraries folder.
Restart the Arduino IDE to ensure the libraries are loaded correctly.
For more information, visit Arduino Library Guide.

1.4 Connecting the Microcontroller and Configuring the Board

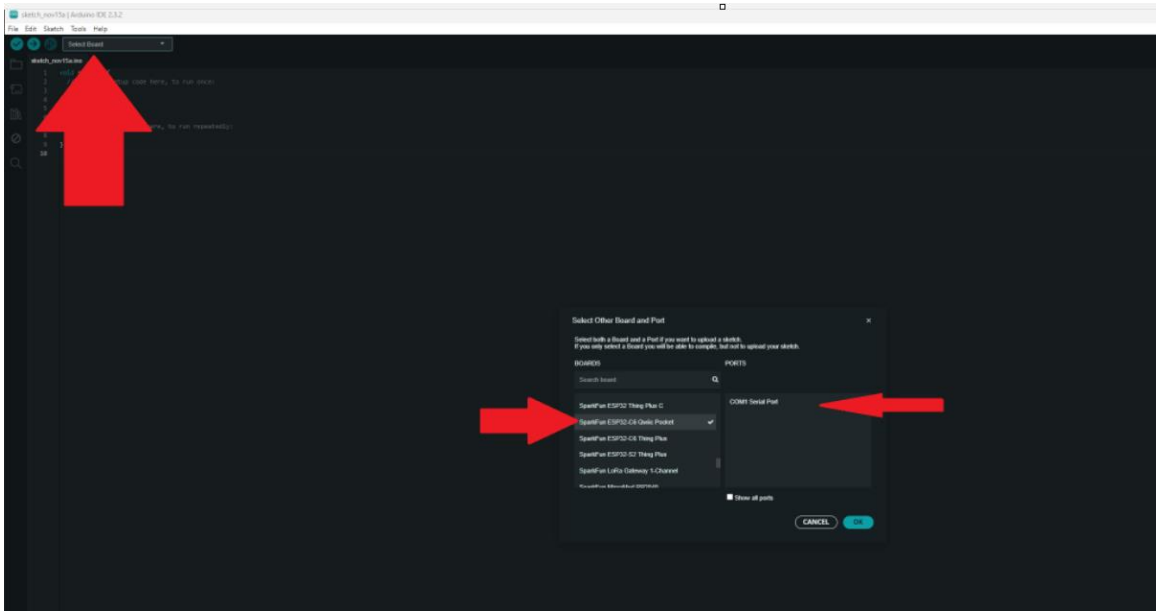
Connecting the Microcontroller and Configuring the Board

Plug your ESP32 or Arduino microcontroller into your computer using a USB cable.

As this is a custom board you will need to import the board config for the board the instructions on doing so are outlined within the ESPRESSIF documentation found at their link: https://docs.espressif.com/projects/arduinoesp32/en/latest/installing.html.

Once completed in the Arduino IDE, go to Tools → Board and select the appropriate board (Sparkfun ESP32 C6 QWIIC) :

- **ESP32**: Choose SparkFun ESP32 QWIIC board.



For detailed instructions on setting up the ESP32, refer to the ESP32 Board Installation Guide.

1.5 Upload Code to the Microcontroller

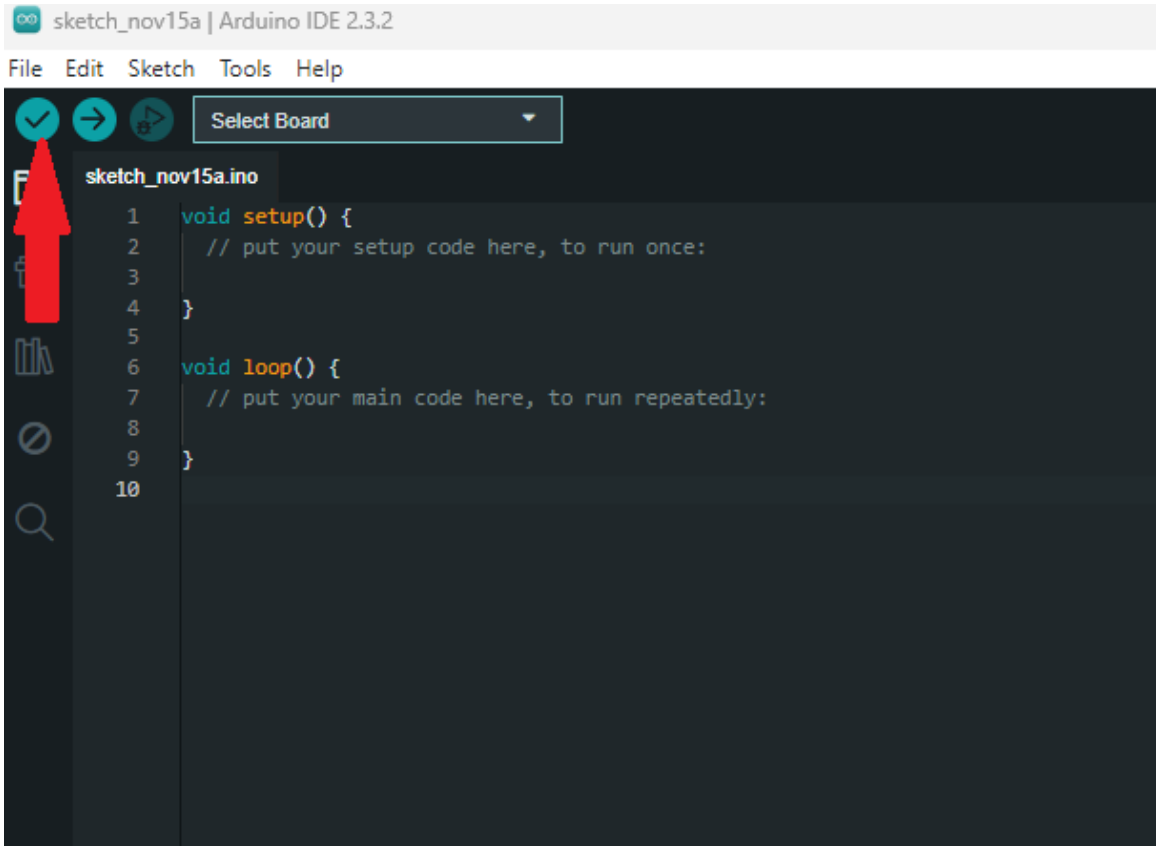
Open your project code in the Arduino IDE.

Verify the code:

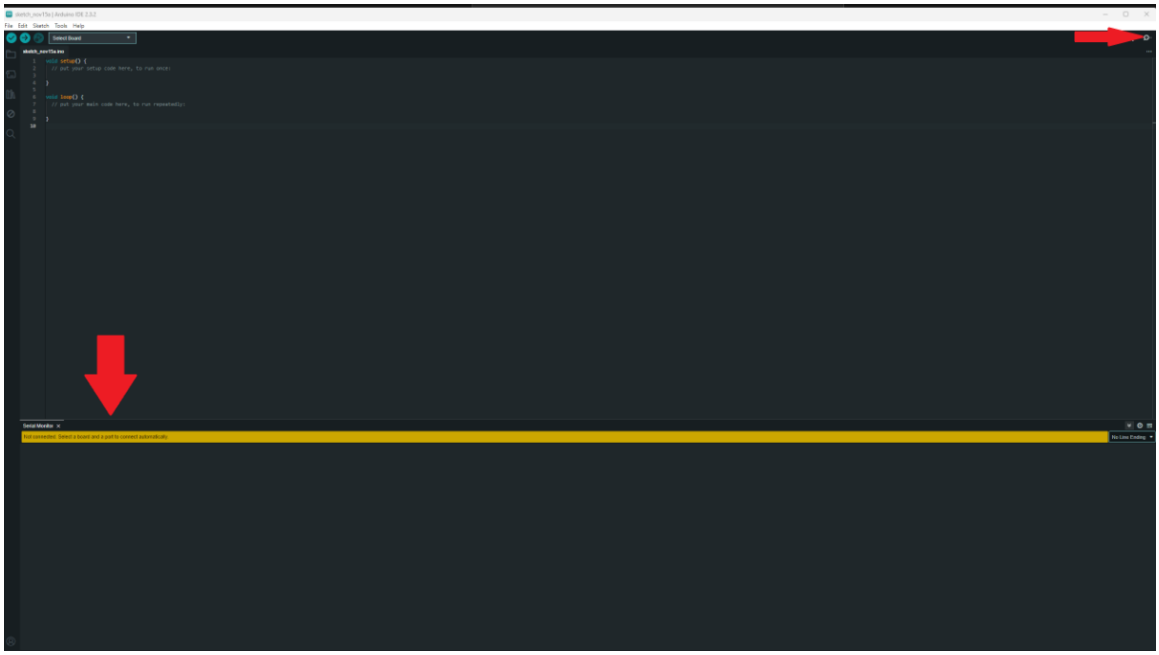
Click the checkmark icon (✓) in the top-left corner of the IDE. This will compile the code to ensure there are no errors.

Upload the code:

Click the right-arrow icon (→) next to the checkmark. The IDE will compile and upload the code to your microcontroller.



Monitor the serial output:
Open the Serial Monitor by navigating to Tools → Serial Monitor to view the realtime data being output by the microcontroller.



For more details on uploading code, refer to Uploading Code to Arduino.

1.6 Troubleshooting Common Issues

If you encounter any issues with library inclusion or code compilation, ensure that:

All libraries are installed correctly.

You have selected the correct board and port.
Your code has no syntax errors.
For additional troubleshooting, refer to Arduino Troubleshooting Guide and the ESPRESSIF troubleshooting guide.

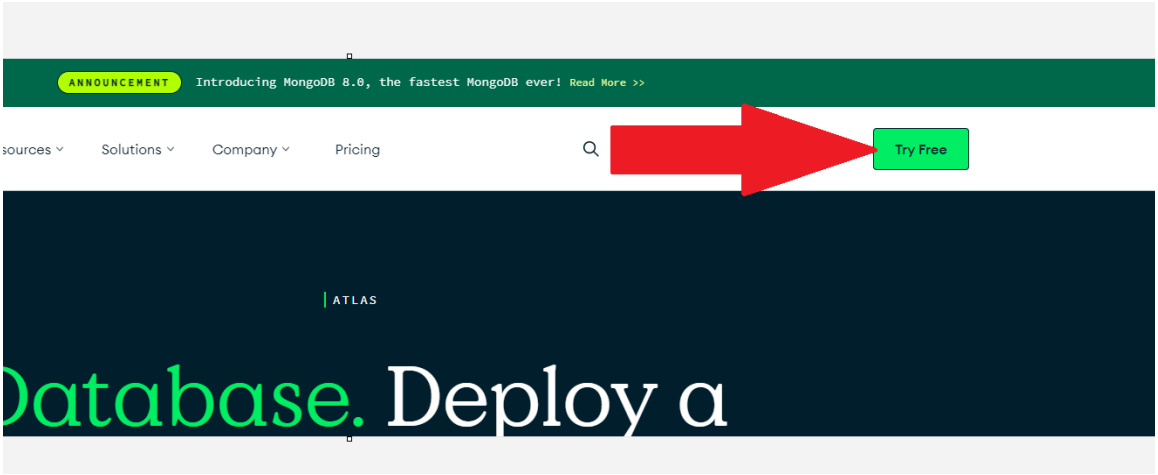
Cloud Integration Documentation

This section covers how to integrate MongoDB for data management.

MongoDB Configuration

Set Up a MongoDB Cluster.

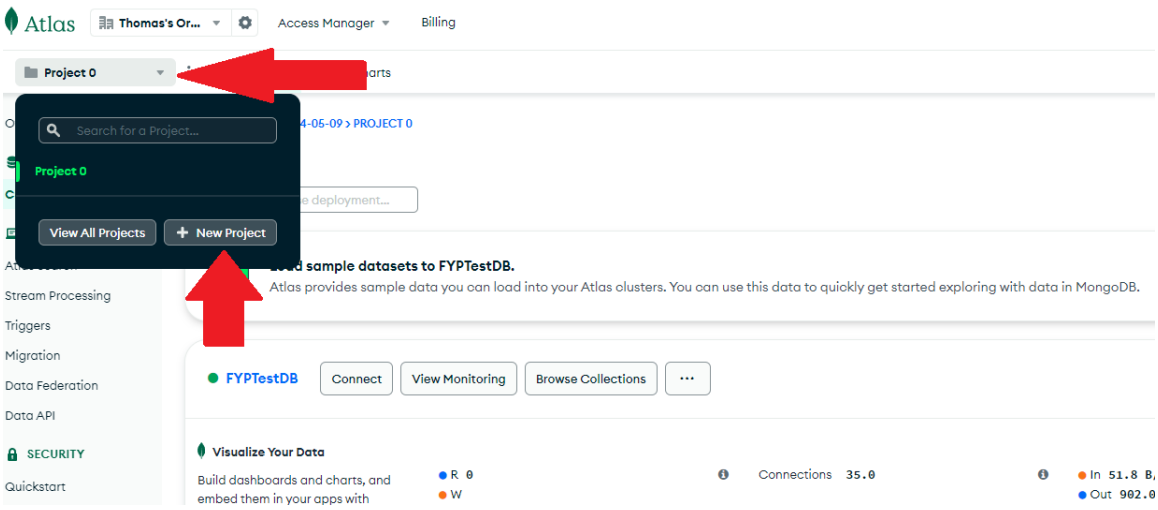
Visit the MongoDB Atlas Website and sign up for a free account.



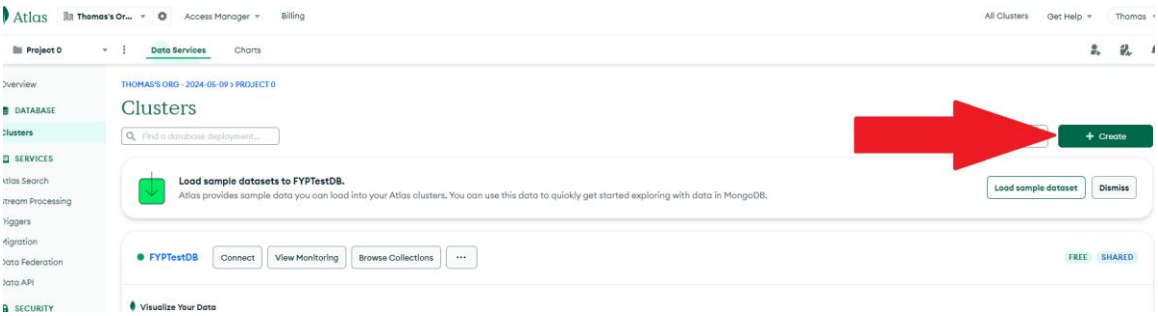
After logging in, create a new project:

Click on the "New Project" button and provide a name for your project.

Click "Create Project" to proceed.

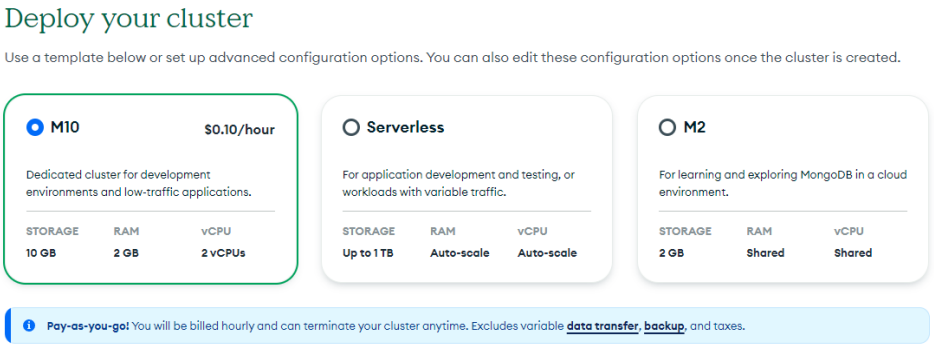


Create a new cluster:

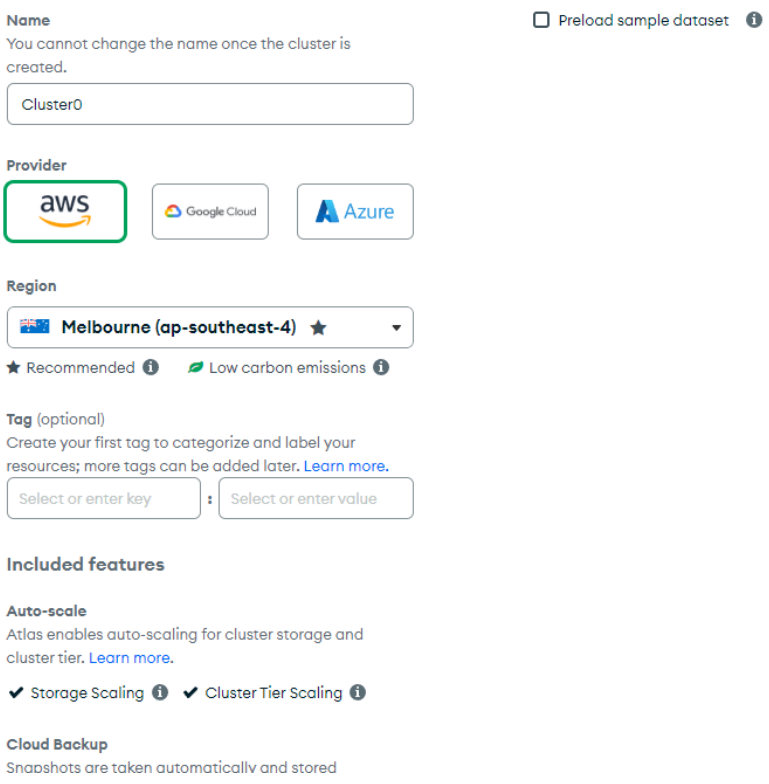


Click on the "Build a Cluster" button.

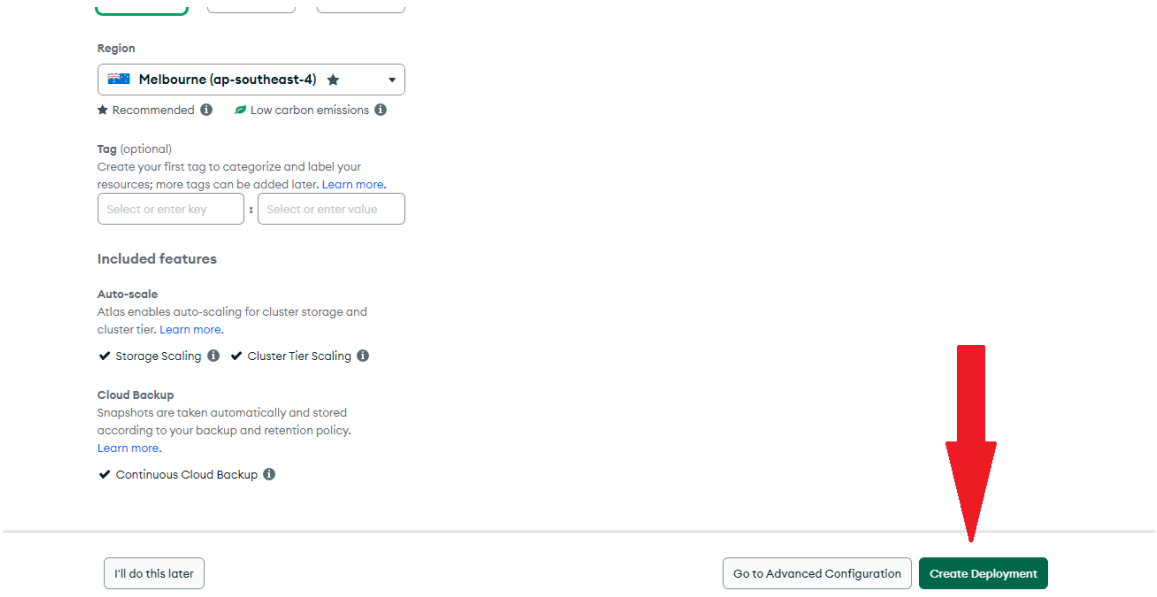
Select the free-tier cluster option for development purposes or your desired cluster type and click "Create Cluster".



Choose a cloud provider (e.g., AWS, Google Cloud, or Azure) and the preferred region.

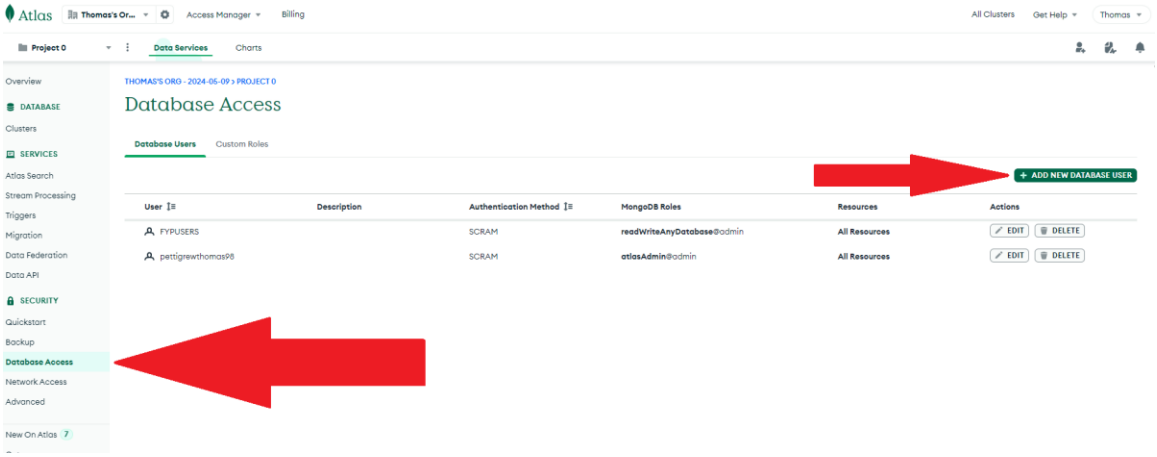


Once the cluster is created, navigate to the cluster dashboard and click "Connect".

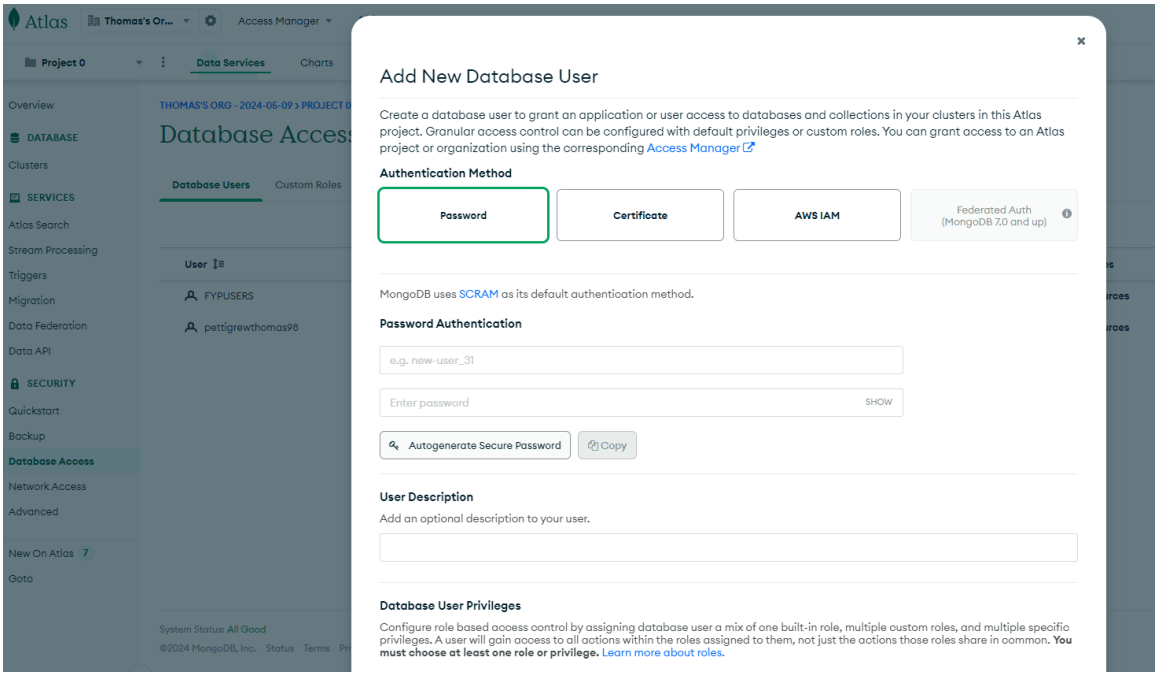


Set up a database user:

Choose "Create a Database User" and provide a username and password.

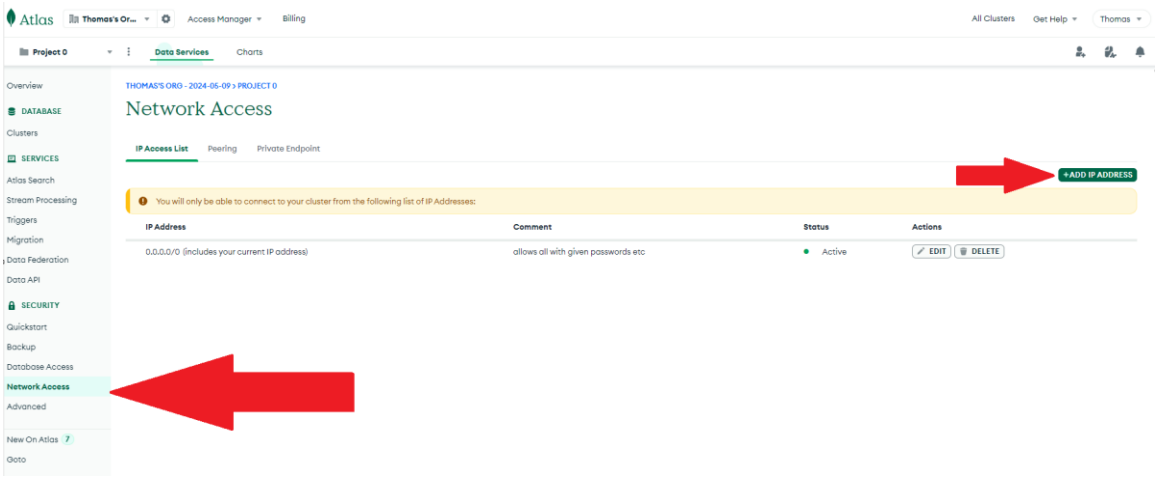


Note down the credentials as they will be needed for connecting the ESP32.



Add your IP address to the IP Whitelist:

Select "Add IP Address" and add your current IP or use 0.0.0.0/0 for unrestricted access (not recommended for production).



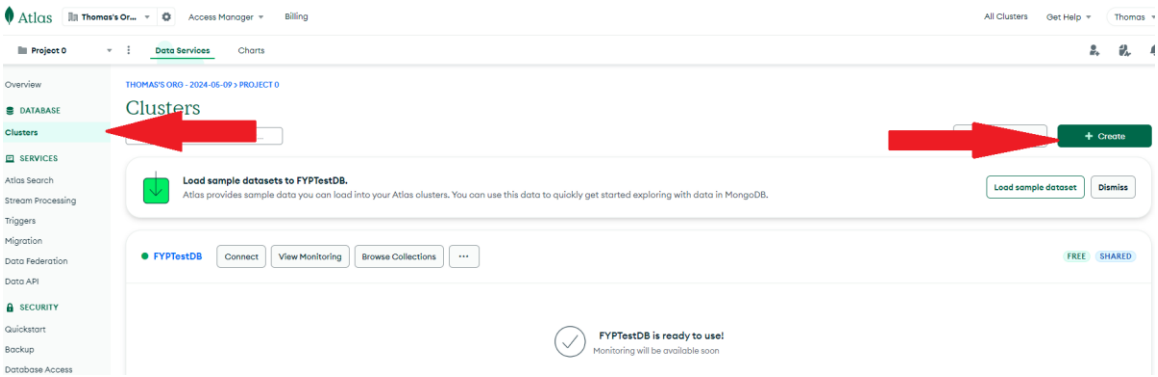
Get the connection string:

Choose "Connect Your Application" and copy the MongoDB URI connection string (e.g., mongodb+srv://<username>:<password>@cluster0.mongodb.net/test).

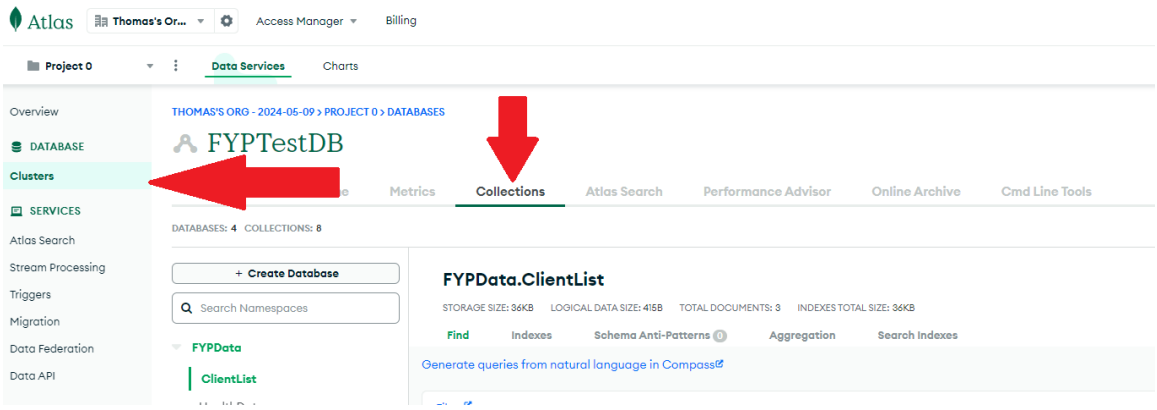
For more information on setting up MongoDB clusters, refer to MongoDB Atlas Documentation.

Create the Database and Collections:

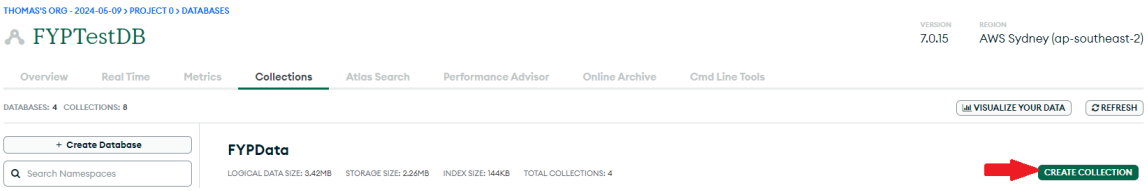
Navigate to your MongoDB Atlas dashboard and click on your cluster.



Open the Collections tab and create a new database by clicking "Add My Own Data".



Provide a database name (e.g., HealthMonitoring) and a collection name (e.g., TuningInformation).



Repeat the process to create additional collections as needed (e.g., HealthData, AutoTuneParameters).

Set Up the MongoDB Database Connection in the Code

Use the MongoDB connection string obtained earlier and insert it into the project's configuration file or source code (e.g., config.h this will be included in the folder with the .ino File).

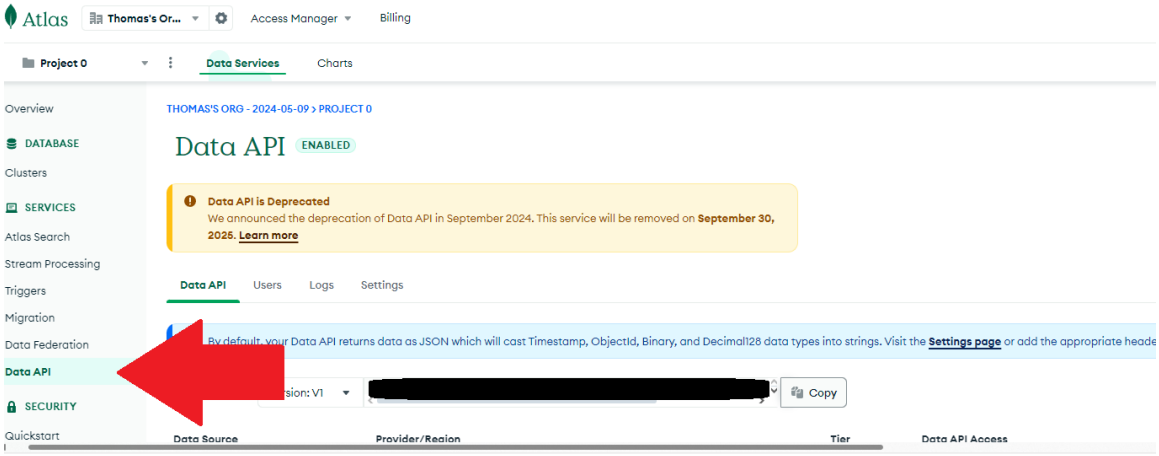
Define the database and collection names as constants to make future changes easier.

For more information on creating databases and collections, refer to [MongoDB Collections and Databases](#).

API Configuration

Create an API key for secure access to MongoDB:

In MongoDB Atlas, select Data API.



Click on "Create API Key" and give it a descriptive name.

Select appropriate permissions, such as "Read and Write Data".
Save the generated API key securely, as it will be needed for setting up connectivity between the ESP32 and MongoDB.

Configure Endpoints
Use the MongoDB connection string as your API endpoint.
Define REST API endpoints within your project code for sending and retrieving data. Here's an example configuration for a POST request to insert data into MongoDB (also found in the config.h):

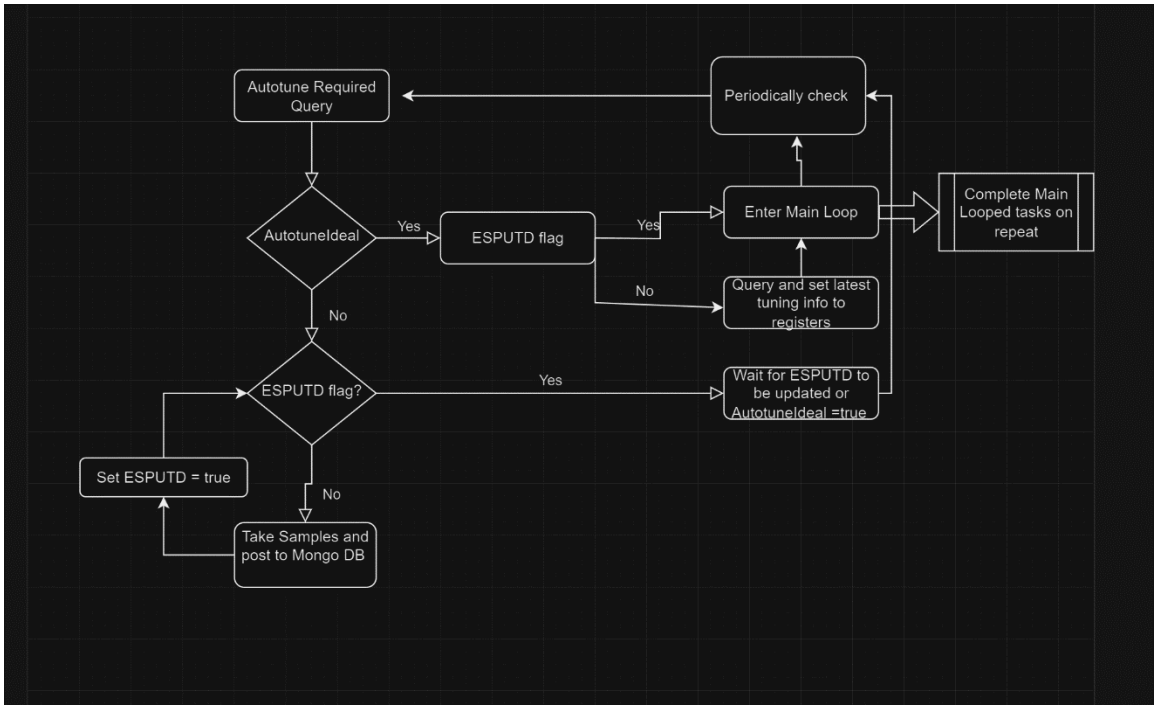
config.h

File Edit View

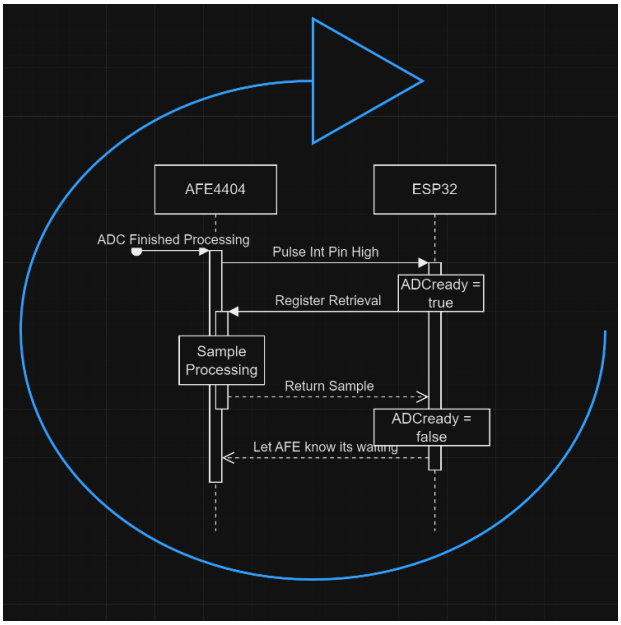
```
// config.h

// Wi-Fi settings
const char* ssid = "[redacted]";
const char* password = "[redacted]";

// MongoDB settings
const char* serverName = "[redacted]";
const char* apiKey = "[redacted]";
const char* deviceID =  ESP32Device2 ";

// Time Server URL
const char* timeServer = "https://www.timeapi.io/api/timezone/zone?timeZone=Australia/Melbourne";
```

Data Flow and Storage

Autotune information flow:



AFE4404 and ESP32 interrupt data flow:



ESP to Mongo Dataflow: