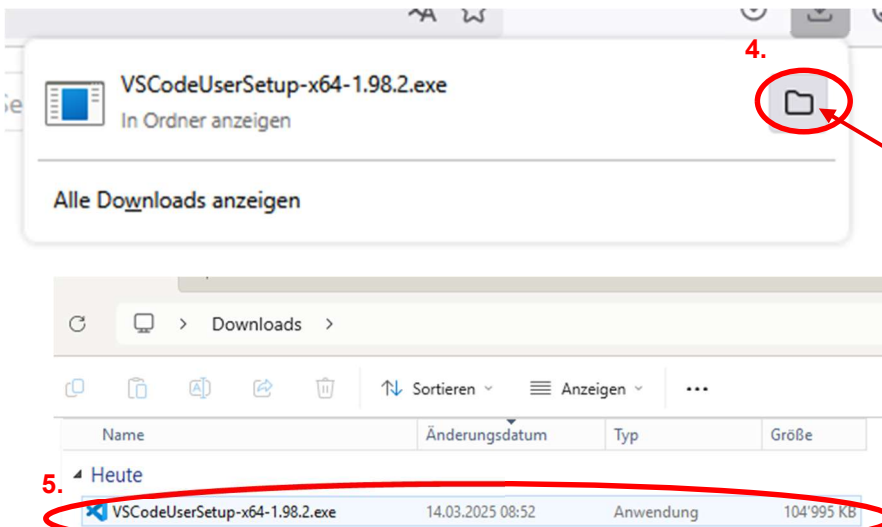
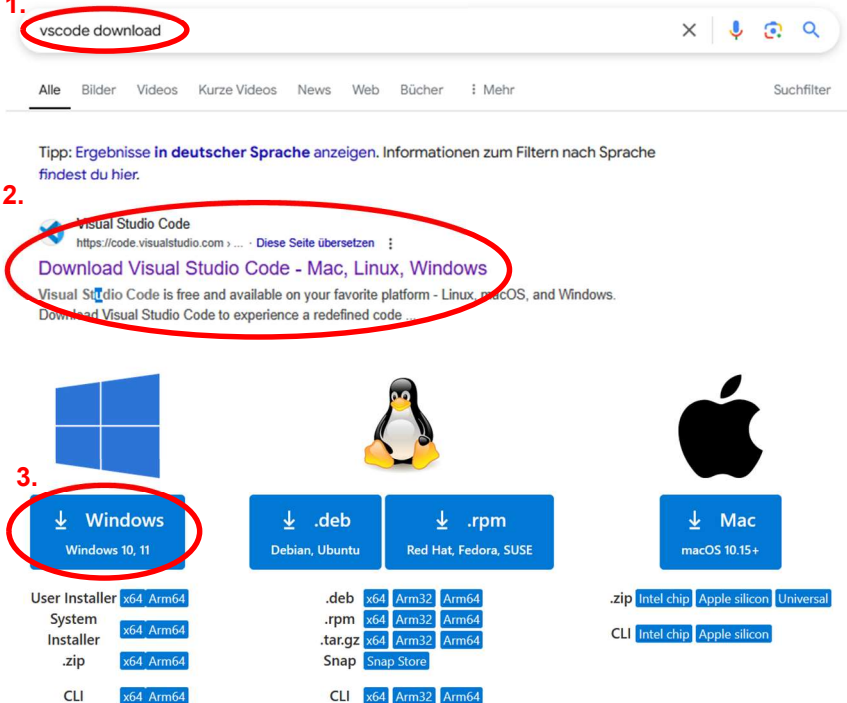
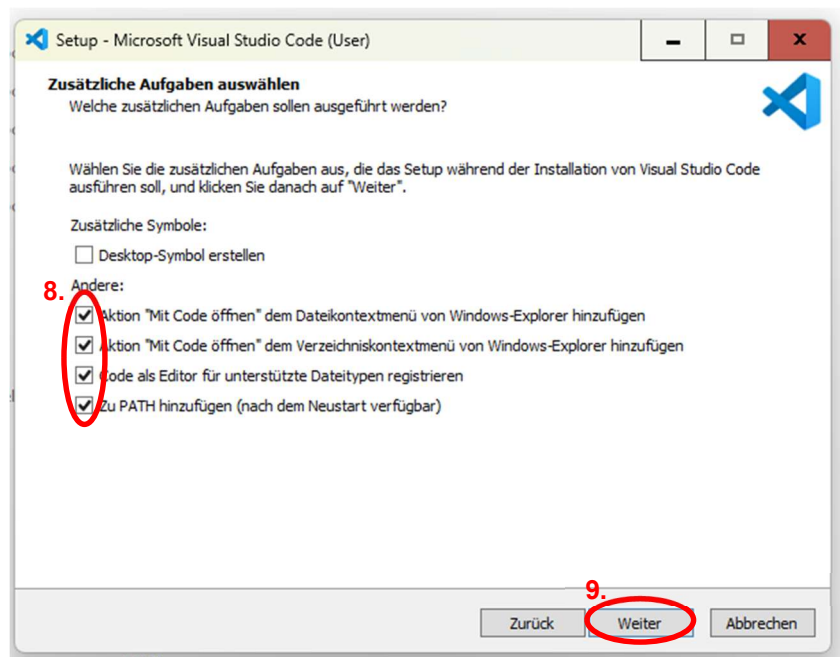
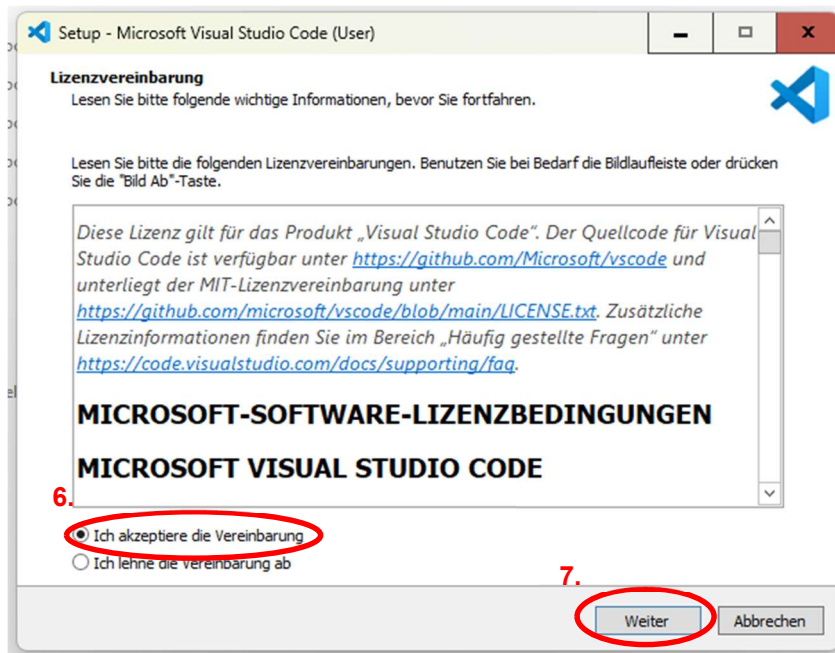


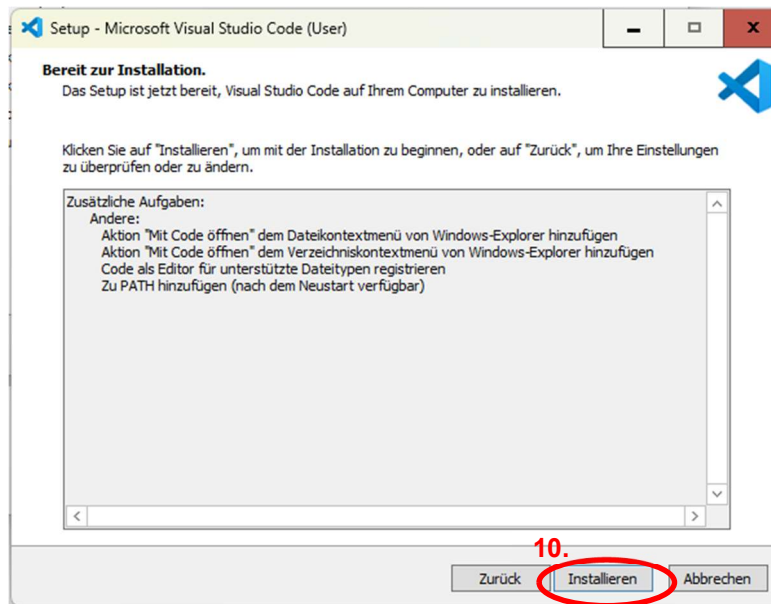
Projekt

Visual Studio Cod manuell einrichten

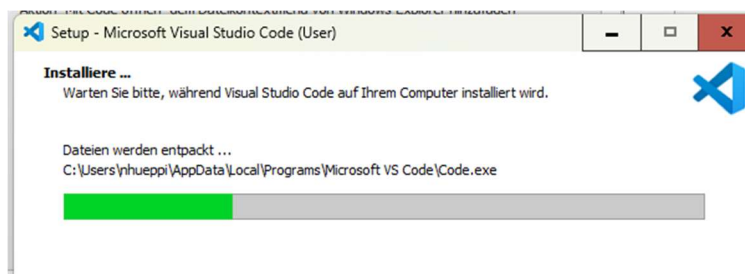
1. **VSCode download** in Google eingeben
2. auf den **rot** umkreisten Link klicken
3. Je nach Betriebssystem herunterladen (in der Anleitung wird Windows genutzt)
4. Warten, bis der Download abgeschlossen ist, und anschließend auf das Ordnersymbol klicken
5. Wenn der Ordner gedrückt wurde dann werden die Downloads im Explorer geöffnet. Die neueste Datei öffnen (VSCodeUserSetup-x64-x.xx.x.exe)



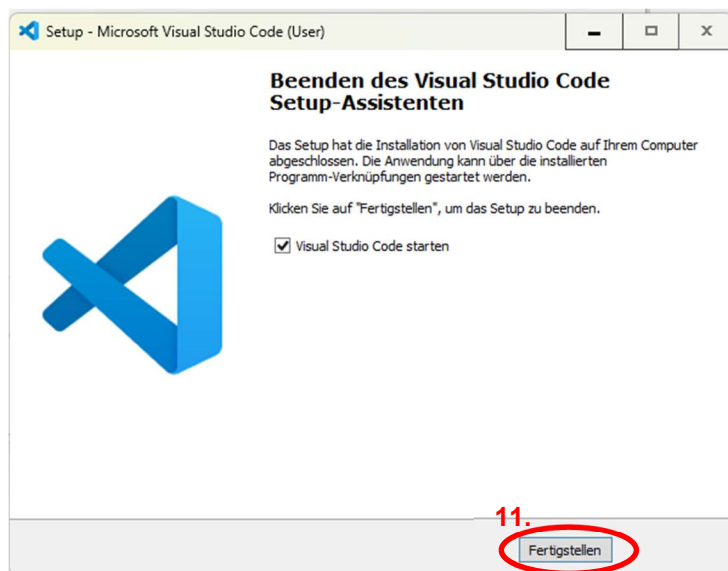




10. Auf **Installieren** drücken



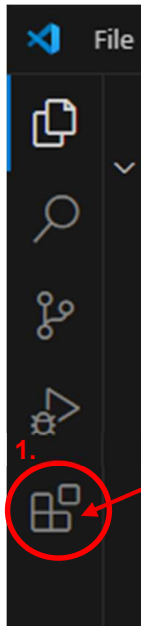
Warten, bis alles installiert ist



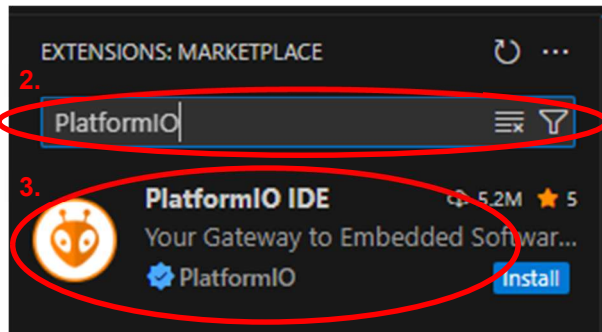
11. Zum Schluss noch auf **Fertigstellen** drücken

Projekt

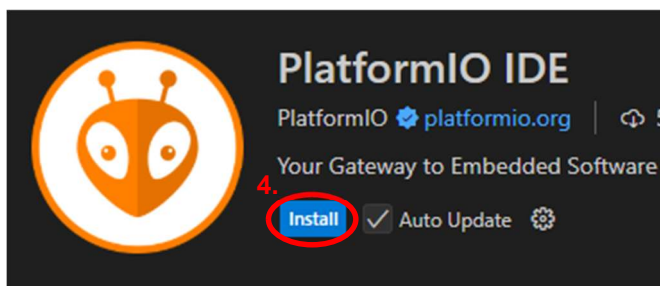
Extension PlatformIO installieren und Projekt erstellen



1. VSCode öffnen.
Bei der Ansichtsauswahl auf Erweiterungen drücken



2. In der Suchleiste **PlatformIO IDE** eingeben
3. Dann die IDE auswählen die einen **orangen Ameisenkopf** hat

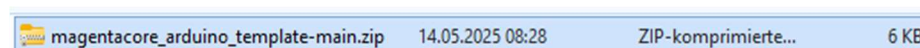
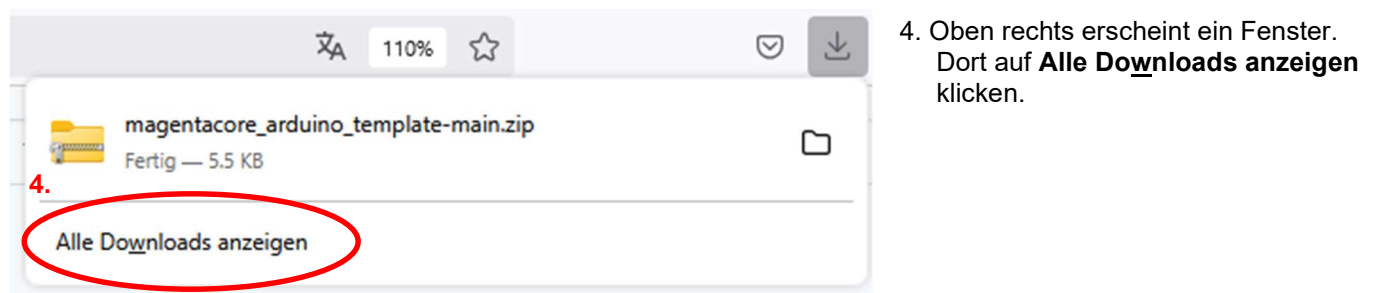
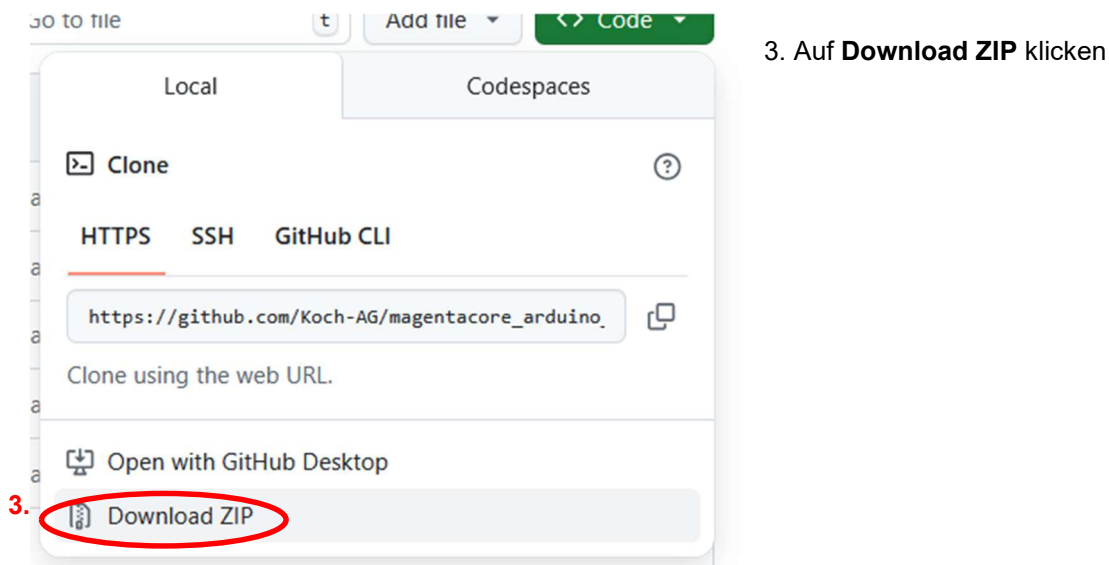


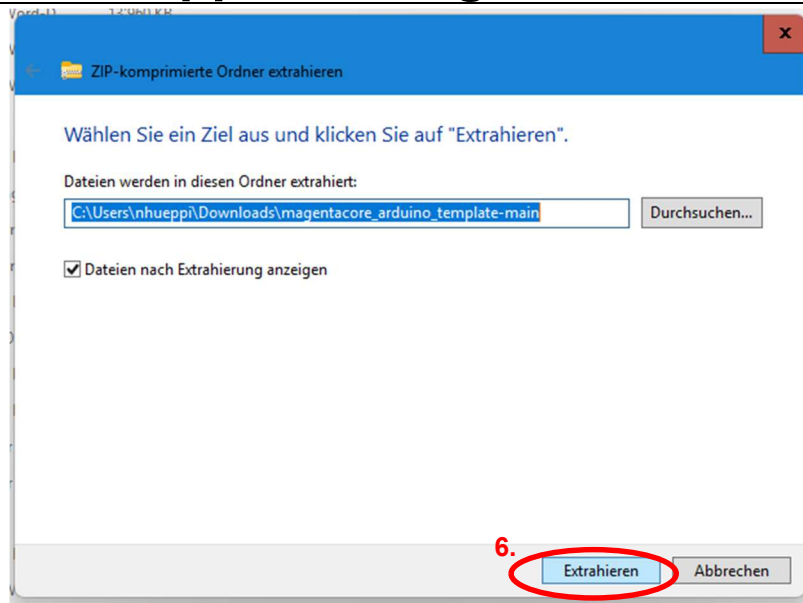
4. auf **Install** drücken

Template herunterladen von Github.com

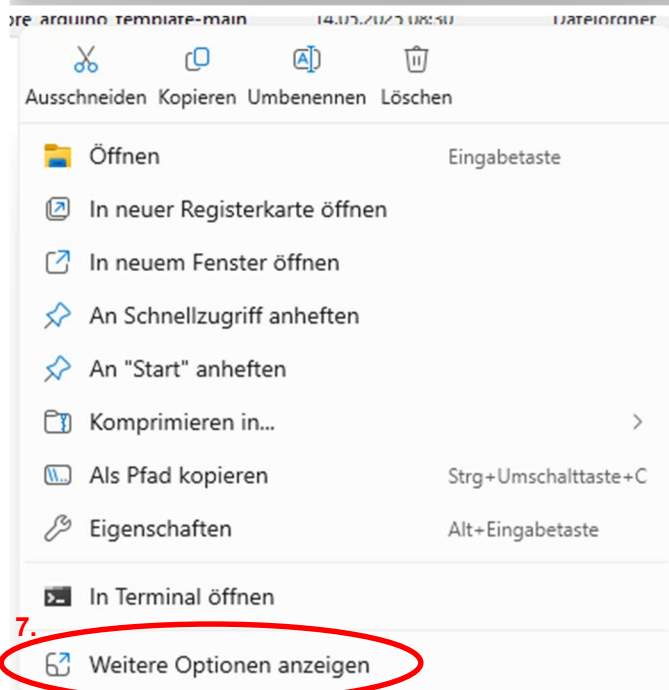
1. Diesen Link: https://github.com/Koch-AG/magentacore_arduino_template in den Browser eingeben.

Wenn man auf die Seite gekommen ist, sollte ein grüner Knopf die Aufmerksamkeit auf sich ziehen. Auf diesem Knopf steht: « < > Code » .

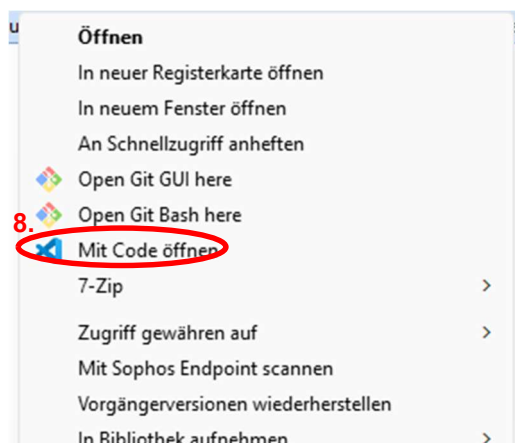




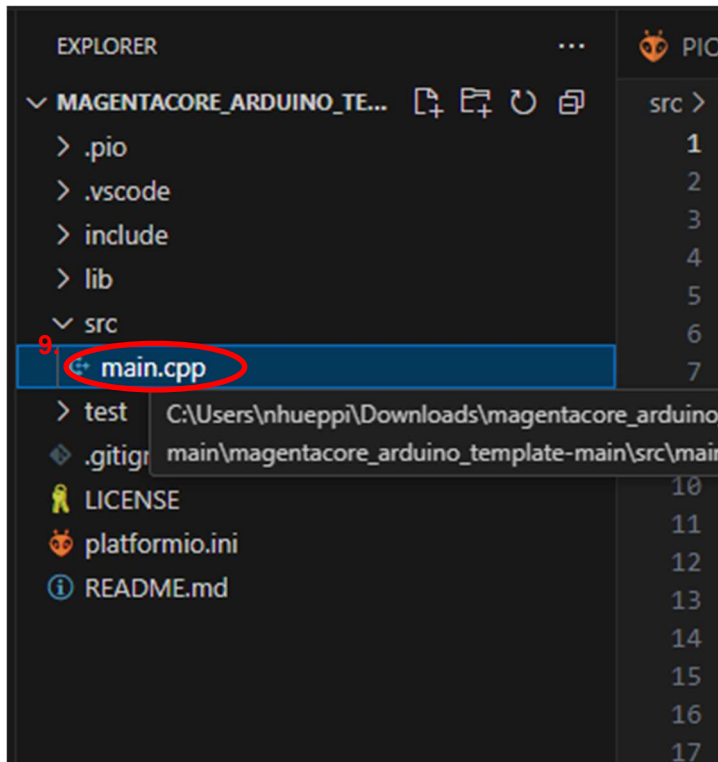
6. Dann auf **Extrahieren** drücken.



7. Es öffnet sich ein Fenster.
Auf **Weitere Optionen anzeigen** drücken.



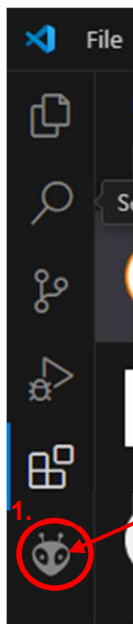
8. Auf **Mit Code öffnen** klicken



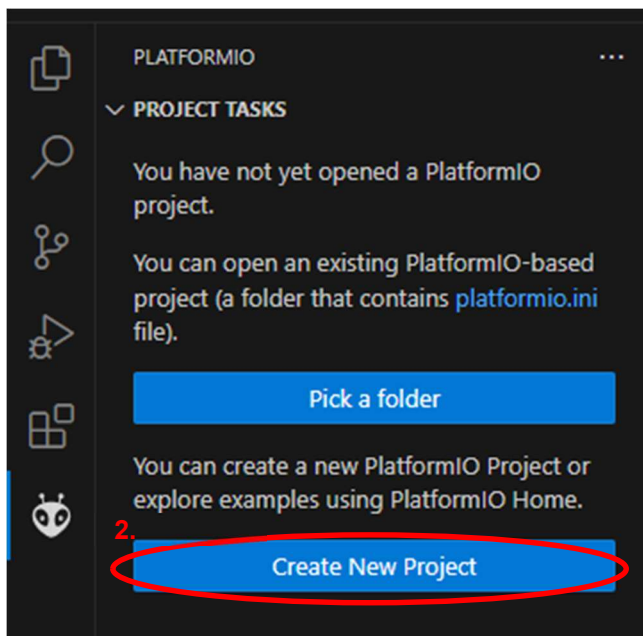
Unten rechts des Bildschirmes wird ein kleines Fenster geöffnet, wo man sieht wie sich PlattformIO einrichtet.

9. Kurz warten.
Dann auf **src** klicken. Unter src **main.cpp** File auswählen.

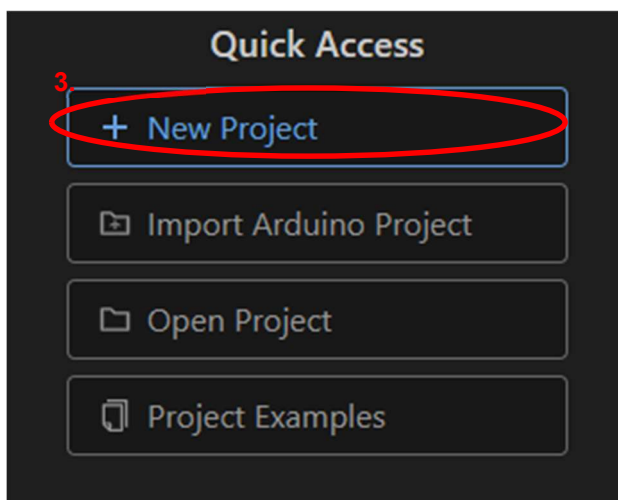
Neues Projekt erstellen



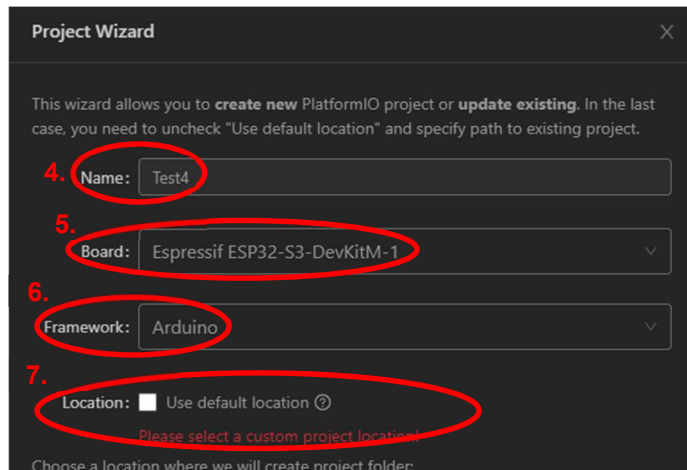
1. Nach der Installation erscheint auf der linken Seite ein Symbol. Das dem Logo der **PlatformIO IDE** entspricht. Dieses Symbol anklicken.



2. Auf **Create New Project** drücken (Falls schon ein Projekt vorhanden ist kann man auf **Pick a folder** Drücken)



3. Auf **New Project** drücken



4. Dem Projekt einen Namen geben

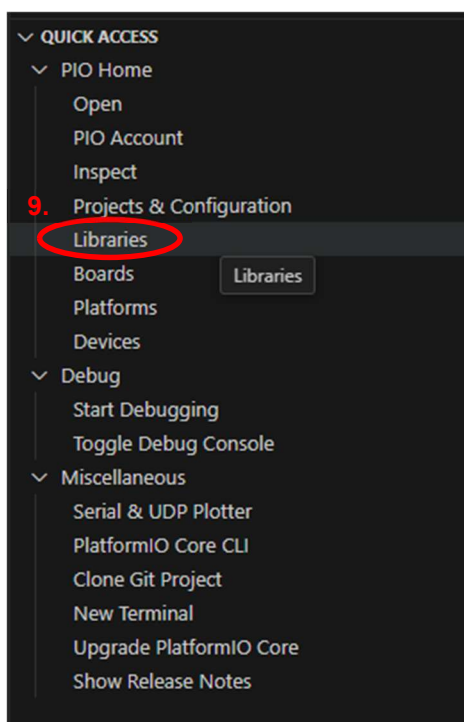
5. Das Board auswählen (**Espressif ESP32-S3-DevKitM-1**)

6. Das Framework angeben (**Arduino**)

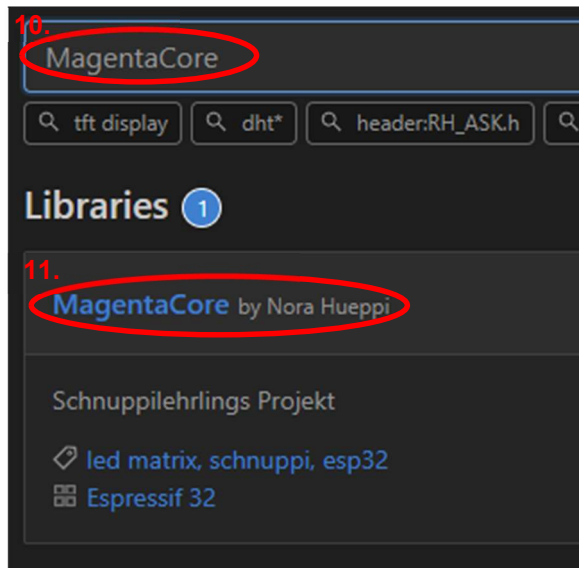
7. Den Haken für Location rausnehmen und dann einen Ordner aussuchen



8. Auf den **Ameisenkopf** drücken

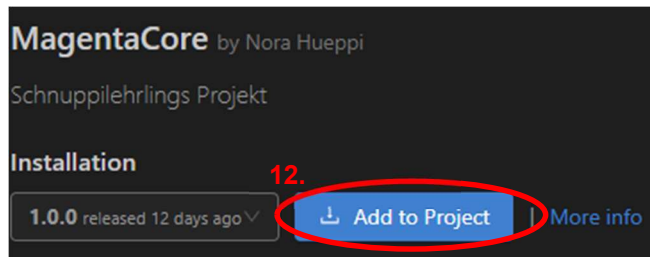


9. Unten hat man mehrere Optionen. Dort auf **Libraries** drücken

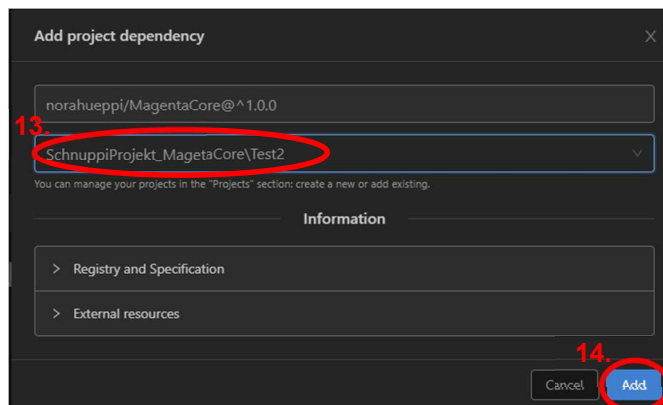


10. In der Suchleiste **MagentaCore** eingeben

11. Die Library von **Nora Hueppi** auswählen.

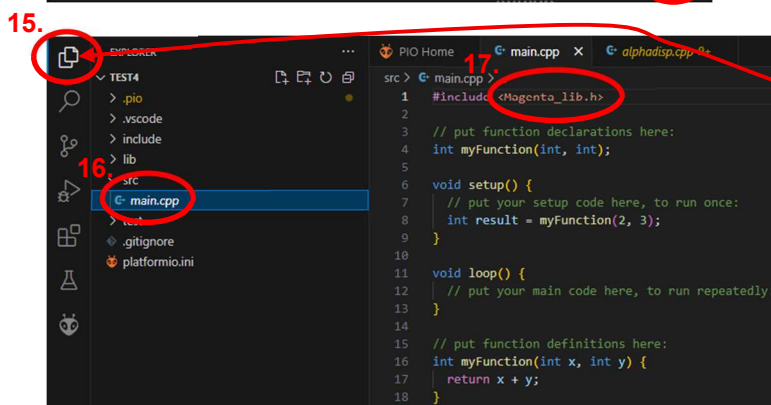


12. Auf **Add to Project** drücken



13. Projekt aussuchen für die Library

14. Projekt aussuchen.
Dann auf **Add** drücken

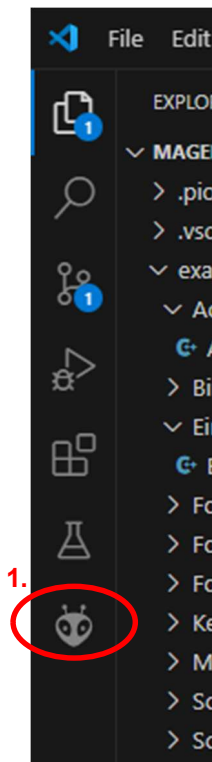


15. Auf die **beiden Blätter** drücken

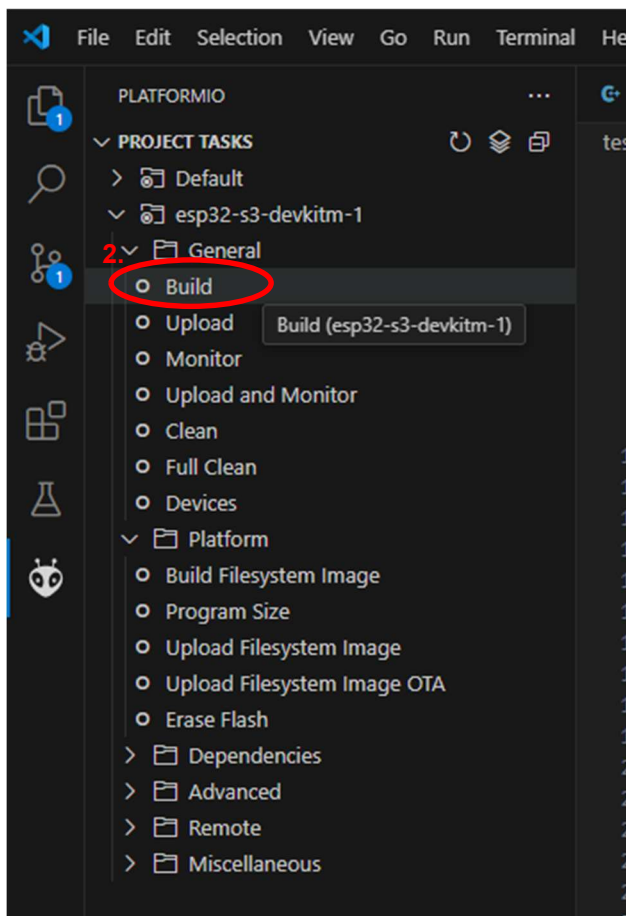
16. Auf **src** (Source) und dann auf **main.cpp** drücken

17. Beim #include <Arduino.h> löschen und #include <Magenta_lib.h> schreiben

Wie Lade ich meinen Code auf das Projekt?



1. auf den **Ameisenkopf** drücken.



2. auf **Build** drücken und warten.

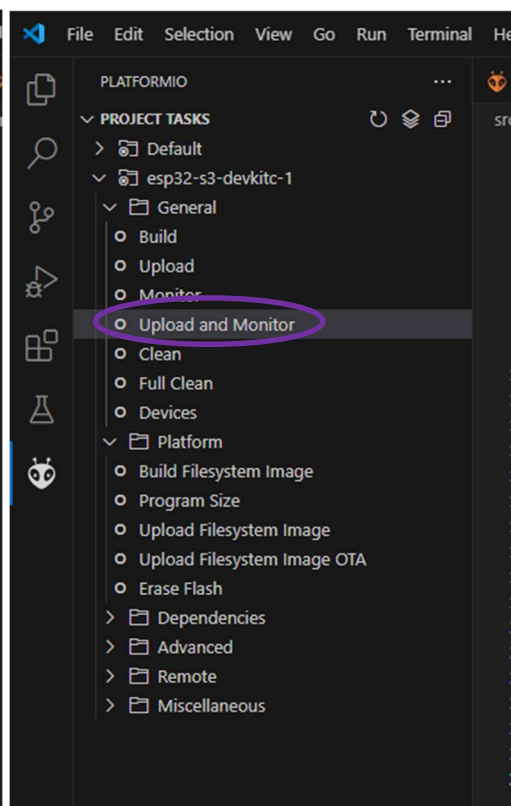
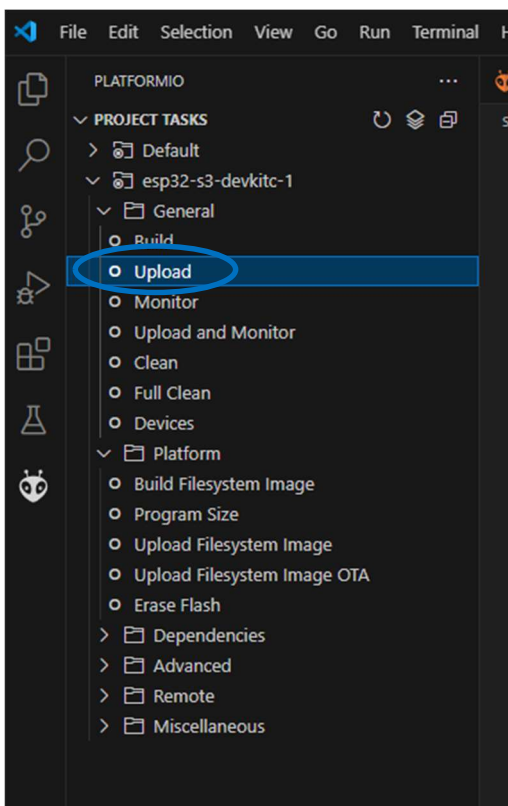
```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

DEBUG: Current (esp-builtin) On-board (esp-builtin) External (cmsis-dap, esp-bridge, esp-prog, iot-bus-jtag, jlink, minimodule, olimex-arm-usb-ocd, olimex-
PACKAGES:
- framework-arduinoespressif32 @ 3.20017.0 (2.0.17)
- tool-esptoolpy @ 1.40501.0 (4.5.1)
- toolchain-riscv32-esp @ 8.4.0+2021r2-patch5
- toolchain-xtensa-esp32s3 @ 8.4.0+2021r2-patch5
LDF: Library Dependency Finder -> https://bit.ly/configure-pio-ldf
LDF Modes: Finder ~ chain, Compatibility ~ soft
Found 39 compatible libraries
Scanning dependencies...
Dependency Graph
|-- MagentaCore @ 1.1.0
Building in release mode
Retrieving maximum program size .pio\build\esp32-s3-devkitc-1\firmware.elf
Checking size .pio\build\esp32-s3-devkitc-1\firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [-] 6.0% (used 19752 bytes from 327680 bytes)
Flash: [-] 9.0% (used 302321 bytes from 3342336 bytes)
===== [SUCCESS] Took 4.09 seconds =====
Terminal will be reused by tasks, press any key to close it.
    
```

Im TERMINAL wird angezeigt, ob es funktioniert hat oder nicht.

Wenn es funktioniert hat dann steht, wie oben im Bild **SUCCESS** und wenn es nicht funktioniert hat, steht **FAILED**. Wenn **FAILED** steht dann hat es einen Fehler im Code und man muss ihn überarbeiten.



Als nächstes kann man entweder auf **Upload** oder **Upload and Monitor** drücken. **Upload and Monitor** ist sinnvoll, wenn man mit einem printf etwas ausgeben möchte.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Hash of data verified.
Compressed 302688 bytes to 169736...
Writing at 0x00010000... (9 %)
Writing at 0x0001bc3d... (18 %)
Writing at 0x0002494f... (27 %)
Writing at 0x00029de6... (36 %)
Writing at 0x0002f2ac... (45 %)
Writing at 0x000348c6... (54 %)
Writing at 0x0003a513... (63 %)
Writing at 0x00042dde... (72 %)
Writing at 0x0004cbfc... (81 %)
Writing at 0x0005206d... (90 %)
Writing at 0x00057be7... (100 %)
Wrote 302688 bytes (169736 compressed) at 0x00010000 in 1.7 seconds (effective 1388.9 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 7.81 seconds =====
Terminal will be reused by tasks, press any key to close it.
    
```

Auch da wird im TERMINAL angezeigt, ob das Hochladen funktioniert hat oder nicht. Dies wird mit **SUCCESS** oder mit **FAIL** angezeigt.