QLock V2.1

Gesamtdokumentation

Architektur · Framework · Module · Konfiguration · Betrieb · Erweiterung

Technisches Projekthandbuch
Stand: 29. Oktober 2026

Entwickler: Klaus Rosenhahn
CO-Entwickler: ChatGPT :)

1 Inhaltsverzeichnis

2 Zweck und Geltungsbereich 4
3 QLock auf einen Blick 5
4 Architektur und Zusammenspiel 6
4.1 Zentrale Steuerung: MCP 6
4.2 Abhängigkeitsübersicht 6
4.3 Typischer Programmablauf 7
5 Verzeichnisse und Dateien 8
6 Display Layout JSON Konfiguration 11
6.1 Display 11
6.2 Fonts – Schriftdefinitionen 12
6.3 Sprachzuordnungen 12
6.3.1 Wochentage 12
6.3.2 Monate 12
6.4 Colors – Farbdefinitionen 13
6.5 Elements – Aufbau der Benutzeroberfläche 14
6.5.1 Wortuhr (wordclock) 14
6.5.2 Informationsbereich (information) 15
6.5.3 Gemeinsame Parameter der Anzeigeelemente 17
6.5.4 Hierarchisches Konfigurationsprinzip 18
6.5.5 10. Abgrenzung zwischen Layout und Programmlogik 18
7 System JSON Konfiguration 19
7.1 Display-Konfiguration (display_layout) 19
7.2 Kalender (calendar) 19
7.3 Wetter (weather) 20
7.4 Sensoren (sensors) 21
7.4.1 Parameter eines Sensoreintrags 21
7.5 Zusammenfassung der Konfigurationsstruktur 22
Grundprinzip 22
8 Framework-Module 23
8.1 framework/logger.py – Logger 23
8.2 framework/timers.py – Timers 25
9 Anwendungs-Module 26
9.1 modules/scheduler.py – Zeitsteuerung 26
9.2 modules/wordclock.py – Wortuhr 26
9.3 modules/calendar.py – Kalender 26
9.4 modules/sensors.py – Sensorverwaltung 26
9.5 modules/weather.py – Wetter und Astronomie 26
9.6 modules/renderer.py – Fenster und Canvas 27
9.7 modules/renderer_display.py – Display-Koordination 27
9.8 modules/renderer_display_wordclock.py – Wortuhr zeichnen 27
9.9 modules/renderer_display_information.py – Informationsanzeige 27
9.10 Unterbereiche der Informationsanzeige 27
10 Konfiguration, Datenquellen und Laufzeitdaten 28
10.1 Datenquellen konfigurieren 28
10.1.1 WeatherAPI 28
10.1.2 Microsoft Calendar vorbereiten 28
10.2 system.json konfigurieren 28
10.3 Layout und Icons 28
10.4 Laufzeitdateien und Logs 29
11 Installation und Inbetriebnahme 30

11.1 Vorbereitung Entwicklungsumgebung Windows 30
11.2 Vorbereitung Entwicklungsumgebung Raspberry 30
11.3 Vorbereitung Datenquellen 30
11.3.1 WeatherAPI 30
11.3.2 Kalender 30
11.3.3 Sensoren 32
11.3.4 Display Layout 32
11.4 Vorbereitung Thonny (Python) 32
11.4.1 Virtuelle Umgebung erstellen 32
11.4.2 Bibliotheken installieren 32
11.5 Erster Start QLOCK 2.0 33
12 Betrieb, Fehlerdiagnose und Wartung 34
12.1 Sinnvolle Reihenfolge bei der Fehlersuche 35
12.2 Logging richtig einsetzen 35
13 Erweiterung und Änderungsregeln 35
13.1 Ein neues Informationsfeld ergänzen 35
13.2 Ein neues Datenmodul ergänzen 35
13.3 Grundregeln für wartbaren Code 36
14 Glossar 36
15 Nächste Änderungen/Neue Version 37
15.1 Abschließende Hinweise 37

2 Zweck und Geltungsbereich

Diese Dokumentation beschreibt QLock V2.1 als zusammenhängendes Softwaresystem: die eigentliche Wordclock-Anwendung, die Darstellung auf dem Bildschirm und das gemeinsam genutzte Framework für Protokollierung, Datenspeicherung, Konfiguration und Zeitfunktionen. Sie dient als technisches Handbuch für Installation, Betrieb, Fehlerdiagnose und Weiterentwicklung.

Grundlage sind die bereitgestellten Python-Module der QLock-Anwendung sowie die drei Framework-Module logger.py, storage.py und timers.py. Die Dokumentation erklärt die erkennbare Architektur und den Programmablauf. Sie ersetzt keinen vollständigen Hardware- und Integrationstest auf dem Ziel-Raspberry-Pi.

> Wichtige Abgrenzung: Im bereitgestellten QLock-ZIP waren die Python-Quellen enthalten, aber nicht alle für einen vollständigen Betrieb nötigen Laufzeitdateien. Insbesondere Konfigurationsdateien, Icons/Bilder und gegebenenfalls Kalender- oder API-Zugangsdaten müssen am tatsächlichen Installationsort vorhanden sein. Wo konkrete Werte nicht aus den gelieferten Dateien hervorgehen, werden sie ausdrücklich als installationsabhängig gekennzeichnet.

3 QLock auf einen Blick

QLock verbindet mehrere Aufgaben in einer Anwendung: Die Wortuhr berechnet die sprachliche Zeitanzeige, der Renderer zeichnet die Inhalte, Sensor- und Wettermodule liefern Messwerte und Wetterinformationen, der Kalender bereitet Termine auf und der Scheduler stellt zeitbezogene Aktualisierungen bereit.

Bereich	Aufgabe	Zentrale Dateien
Steuerung	Module erzeugen, starten, zyklisch bedienen, stoppen und Fehler abfangen	mcp.py
Framework	Logging, JSON-Konfiguration/Runtimedaten, Zeit-Hilfsfunktionen	framework/logger.py, storage.py, timers.py
Zeit und Anzeigeinhalt	Wortclock-Matrix und zeitabhängige Aktualisierung	modules/wordclock.py, scheduler.py
Datenquellen	Sensorwerte und Wetterdaten abrufen/aufbereiten	modules/sensors.py, weather.py
Kalender	ICS-Kalender laden, wiederkehrende Termine berücksichtigen und sortieren	modules/calendar.py
Bildschirm	Tkinter-Fenster, Layout und einzelne Anzeigebereiche zeichnen	modules/renderer*.py

Das Design trennt fachliche Aufgaben von der Darstellung: Sensoren müssen nicht wissen, an welcher Bildschirmposition ihre Werte erscheinen; die Anzeige wiederum soll Messwerte aus dem zentralen Laufzeitstatus lesen, statt selbst Datenquellen abzufragen.

4 Architektur und Zusammenspiel

4.1 Zentrale Steuerung: MCP

mcp.py enthält die Klasse MCP. Sie erstellt zunächst den gemeinsamen Modulcontainer und stellt zentrale Dienste bereit. Danach werden die Fachmodule angelegt. Die Reihenfolge ist wichtig: Ein Modul kann beim Erzeugen nur auf Module zugreifen, die bereits im Container registriert sind.

Der MCP verwaltet den Lebenszyklus über begin(), service() und stop(). begin() ruft die Startmethode der registrierten Module auf. service() durchläuft die Module und ruft – sofern vorhanden – deren service()-Methode auf. Ausnahmen werden pro Modul abgefangen und über die zentrale Fehlerbehandlung protokolliert, damit ein einzelner Fehler nicht zwangsläufig den gesamten Durchlauf abbricht.

4.2 Abhängigkeitsübersicht

Modul	Benötigt / greift zu auf	Liefert
Logger	Dateisystem und Zeit	Modulbezogene Logmeldungen und Fehlerprotokolle
Timers	Lokale Systemzeit	Zeit-, Datums- und Intervallfunktionen
Storage	MCP/Dateisystem	system-Konfiguration und Laufzeitdaten
Scheduler	Timers, Storage und Anwendungszustand	Zeit-/Minutenwechsel- und Erinnerungszustand
Calendar	Storage, HTTP/ICS-Bibliotheken	Aufbereitete Kalendertermine
Sensors	Storage, HTTP-Anfragen und Weather-Modul	Innen-/Außen-Sensorwerte
Weather	Storage, HTTP/API und Icons	Wetter-, Wind-, Astro- und Icon-Daten
WordClock	Storage und Konfiguration	Aktuelle Wörter und Matrixzustand
Renderer	Tkinter, Layout und Display-Renderer	Sichtbare Anzeige und Aktualisierung

4.3 Typischer Programmablauf

1. Start: MCP wird erstellt und registriert Logger, Timers, Storage sowie die Anwendungs-Module.
2. Initialisierung: begin() wird auf den Modulen aufgerufen. Dateien, Konfigurationen, Datenquellen und Anzeigeelemente werden vorbereitet.
3. Laufender Betrieb: Die Hauptschleife ruft MCP.service() regelmäßig auf. Jedes Modul erledigt dabei nur den jeweils fälligen kurzen Arbeitsschritt.
4. Datenaufbereitung: Sensor-, Wetter-, Kalender- und Zeitmodule aktualisieren ihren Zustand.
5. Darstellung: Der Renderer liest die aktuellen Zustände der Arbeitsmodule und zeichnet die Wortuhr und Informationsbereiche.
6. Fehlerfall: Der MCP protokolliert Ausnahmen mit Modulkontext; die Anwendung kann den nächsten Durchlauf fortsetzen.
7. Beenden: stop() wird in umgekehrter Modulreihenfolge aufgerufen, soweit ein Modul eine Stop-Methode besitzt.

5 Verzeichnisse und Dateien

Verzeichnis / Dateiname	Aufgabe
./_Arduino	Software für Sensoren. Separate Dokumentation, wird hier nicht beschrieben
./_Dokumentation	Ablage Dokumentation. Auch diese hier
./_Environment	Das virtuelle Environment von Python hier installieren
./_Installation	Installations-Scripte für Python
./configuration	Die Konfigurationdaten für physisches Display und QLock-Layout
./configuration/Display/	Daten für diverse Display-Größen
./configuration/Display/1024x768.json	Konfiguration für 1024x768 Pixel. Siehe Kapitel “Display Layout”
./configuration/Display/1024x720.json	Konfiguration für 1024x720 Pixel. Siehe Kapitel “Display Layout”
./configuration/Display/1024x800.json	Konfiguration für 1024x800 Pixel. Siehe Kapitel “Display Layout”
./configuration/runtime.json	wird nicht ausgeliefert, sondern während Laufzeit erzeugt
./configuration/system.json	Wichtige Systemparameter. Vorher prüfen und aktualisieren. Siehe Kapitel “system.json”
./configuration/wordmatrix.json	Die Wortmatrix und Wortsequenzen
./framework	Die Basismodule für jedes Python-Projekt. Siehe “Framework-Module”
./framework/logger.py	Aufbereitung und Formatierung von Fehlermeldungen. Schreibt Log-Informationen in Direktfenster und Log-Dateien
./framework/storage.py	Speicherverwaltung für .json und Laufzeitdaten
./framework/timers.py	Bereitstellung von formatierten Zeiten und Taktgeber

./icons	Ablage verwendeter Grafikobjekte. Achtung: ./icons/images ist Display-abhängig (noch)
./log	Ablage der Log-Dateien. Wird zyklisch bereinigt
./modules	Alle Hilfsmodule zur Darstellung und Berechnung der Ausgabe. Siehe “Anwendungs-Module”
./modules/calendar.py	Verarbeitung und Bereitstellung von Kalenderdaten. Siehe “calendar.py – Kalender”
./modules/renderer.py	Verarbeitung und Bereitstellung von Ausgabemodulen
./modules/renderer_display.py	Koordination der Ausgabemodule auf dem Display
./modules/renderer_display_information.py	Ausgabemodul für Informations-Panel (verwaltet Untermodule)
./modules/renderer_display_information_calendar.py	Ausgabemodul für Termine
./modules/renderer_display_information_date.py	Ausgabemodul für Datum
./modules/renderer_display_information_temperature.py	Ausgabemodul für Temperaturen innen/aussen
./modules/renderer_display_information_temperature_buttons.py	Ausgabemodul für UmschaltknöpfeTemperatur
./modules/renderer_display_information_weather.py	Ausgabemodul für Sonnenauf-/untergang und Wetter-Icon
./modules/renderer_display_information_wind_moon.py	Ausgabemodul für Wind, Mondauf-/untergang
./modules/renderer_display_wordclock.py	Ausgabemodul für Wortuhr
./modules/scheduler.py	Verarbeitung und Bereitstellung zeitgesteuerter Ereignisse
./modules/sensors.py	Verarbeitung und Bereitstellung von Sensorendaten
./modules/weather.py	Verarbeitung und Bereitstellung von Wetterdaten
./modules/wordclock.py	Verarbeitung und Bereitstellung von Zeitdaten für die Wortuhr
./mcp.py	Master Control Programm :) startet und verwaltet Module. Enthält keine Fachlogik.

6 Display Layout JSON Konfiguration

Die Display-Konfiguration beschreibt das Layout und die Darstellung der Benutzeroberfläche von QLock. Sie legt die Bildschirmabmessungen, Schriftarten, Farben, sprachabhängige Bezeichnungen sowie die Positionierung und Gestaltung der einzelnen Anzeigeelemente fest.

Die Konfiguration ist hierarchisch aufgebaut und wird im JSON-Format gespeichert.

Ziel ist es, die Darstellung der Benutzeroberfläche möglichst weitgehend durch Konfigurationsdaten zu steuern, ohne dafür Änderungen am Programmcode vornehmen zu müssen.

Die Konfiguration ist in folgende Hauptbereiche gegliedert:

Bereich	Funktion
Display	Festlegung der Bildschirmabmessungen
Fonts	Definition der verfügbaren Schriftarten und Schriftgrößen
Weekday Mapping	Zuordnung der Wochentage
Month Mapping	Zuordnung der Monatsnamen
Colors	Definition der verwendeten Farben
Elements	Beschreibung der Anzeigebereiche und ihrer Inhalte

6.1 Display

Der Bereich auf oberster Ebene definiert die Abmessungen der Benutzeroberfläche.

Parameter	Bedeutung	Beispiel
width	Bildschirmbreite in Pixeln	1280
height	Bildschirmhöhe in Pixeln	800

Die angegebenen Werte bilden das Koordinatensystem für die Positionierung der Anzeigeelemente. Das hier beschriebene Layout hat eine Auflösung von 1280 × 800 Pixeln. In der ausgelieferten system.json ist derzeit jedoch das Layout 1280x720 voreingestellt. Es handelt sich um unterschiedliche Layoutvarianten; die ausgewählte Layoutdatei und die Geometrie ihrer Elemente müssen zur tatsächlich verwendeten Auflösung passen.

Hinweis: Für die Inbetriebnahme ist der Wert display_layout.layout in system.json maßgeblich. Die Auflösung in der Layoutdatei sollte mit den Abmessungen des verwendeten Fensters beziehungsweise Displays übereinstimmen.

6.2 Fonts – Schriftdefinitionen

Der Bereich fonts definiert die verfügbaren Schriftarten. Die Schriftdefinitionen werden über symbolische Namen referenziert.

Jede Schriftdefinition enthält folgende Parameter:

Parameter	Bedeutung
family	Name der Schriftfamilie
size	Schriftgröße
weight	Schriftstärke, beispielsweise normal oder bold

Vordefinierte Schriftklassen:

Bezeichnung	Schriftgröße	Verwendung
XXL	50	Sehr große Schrift
XL	40	Extra große Schrift
L	30	Große Schrift
M	20	Mittlere Schrift
S	15	Kleine Schrift
XS	10	Sehr kleine Schrift
letter	25	Buchstabenmatrix der Wortuhr, fett

Die Schriftklassen ermöglichen eine einheitliche Gestaltung. Die einzelnen Anzeigeelemente referenzieren die gewünschte Schrift über den Parameter font. Eigene Bezeichner und Font-Einstellungen können verwendet und das Schriftsystem erweitert werden.

6.3 Sprachzuordnungen

6.3.1 Wochentage

Der Bereich weekday_mapping enthält die ausgeschriebenen Namen der sieben Wochentage.
Die Reihenfolge beginnt mit Montag und endet mit Sonntag.

6.3.2 Monate

Der Bereich month_mapping enthält die Namen der zwölf Monate in chronologischer Reihenfolge, beginnend mit Januar.
Beide Bereiche dienen der sprachgerechten Darstellung von Datumsinformationen.

6.4 Colors – Farbdefinitionen

Der Bereich colors definiert die Farben für unterschiedliche Betriebs- und Darstellungszustände.

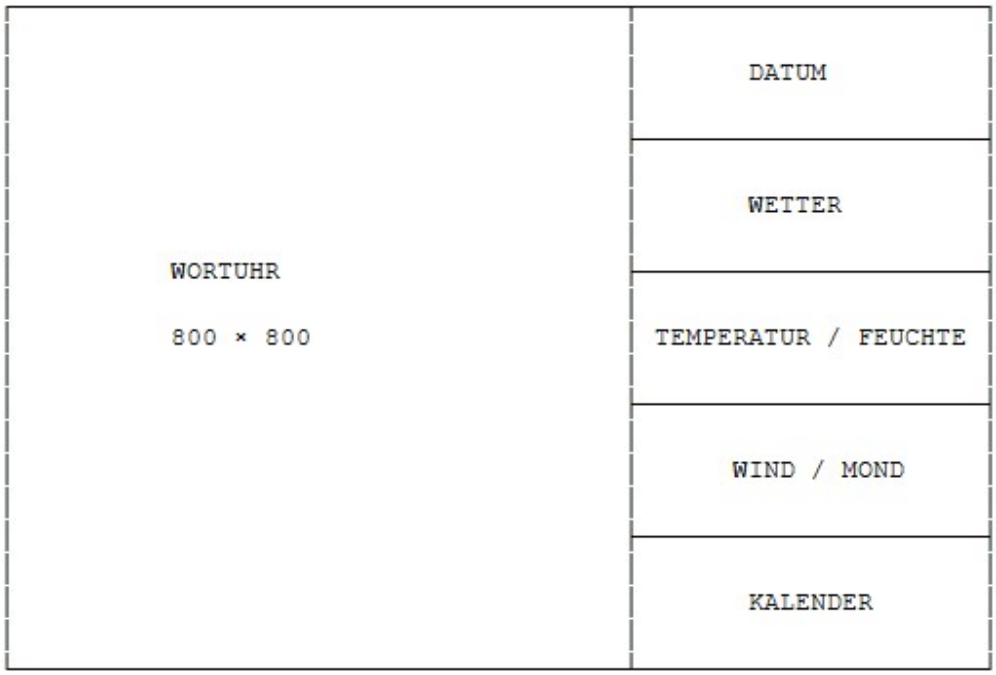
Schlüssel	Bedeutung	Farbe
day	Farbdefinition für den Tagmodus	Weiß
1	Alternative beziehungsweise numerische Zuordnung	Weiß
night	Farbdefinition für den Nachtmodus	Bernsteinfarben (#FFBF00)
0	Alternative beziehungsweise numerische Zuordnung	Bernsteinfarben (#FFBF00)
inactive	Farbe für inaktive Elemente	Dunkelgrau (#303030)

Die Konfiguration sieht damit eine helle Darstellung im Tagmodus und eine bernsteinfarbene Darstellung im Nachtmodus vor. Die konkrete Zuordnung der Schlüssel zu den Betriebszuständen erfolgt durch den Programmcode.

6.5 Elements – Aufbau der Benutzeroberfläche

Der Bereich elements beschreibt die räumliche Struktur der gesamten Benutzeroberfläche.
Die Anzeige ist in zwei Hauptbereiche gegliedert:

- Wortuhr
  Die Wortuhr befindet sich auf der linken Bildschirmseite. Die Maße sind so gewählt, dass die Wortuhr einen quadratischen Raum einnimmt.
- Informationsbereich
  Den verbleibenden Rest des Display füllt der Informationsbereich mit den Sektionen Datum, Wetter, Temperatur/Feuchte, Wind/Mond und Kalender



6.5.1 Wortuhr (wordclock)

Parameter	Wert
X-Position	0
Y-Position	0
Breite	800 Pixel
Höhe	800 Pixel
Schriftklasse	letter
Skalierungsfaktor	0,85

Zusätzlich werden der horizontale und vertikale Buchstabenabstand über letter_spacing_x und letter_spacing_y konfiguriert.

Der Unterbereich minute_dots definiert die Darstellung der Minutenpunkte über einen Radiusfaktor und einen Randabstandsfaktor.

6.5.2 Informationsbereich (information)

Der Informationsbereich befindet sich rechts neben der Wortuhr.

Parameter	Wert
X-Position	800
Y-Position	0
Breite	480 Pixel
Höhe	800 Pixel

Der Bereich enthält eine Rahmenkonfiguration sowie eine Liste horizontaler Trennlinien.
Jede Trennlinie wird über ihre Y-Position und den Parameter visible beschrieben.
Der Informationsbereich ist in folgende Inhaltsbereiche unterteilt:

- Datum
- Wetter
- Temperatur und Luftfeuchtigkeit
- Wind und Mond
- Kalender

6.5.2.1 Datum (date)

Der Datumsbereich enthält zwei Anzeigeelemente:

Element	Funktion	Schriftklasse
weekday	Anzeige des Wochentags	L
date	Anzeige des Datums	L

Die Positionierung erfolgt über Y-Koordinaten, Höhenangaben und den Textanker.

6.5.2.2 Wetter (weather)

Der Wetterbereich enthält Informationen zum Sonnenaufgang, Sonnenuntergang, Wetterzustand und Luftdruck.

Element	Funktion
sunrise	Sonnenaufgang
sunset	Sonnenuntergang
icon_cloud	Wetter-Icon
pressure	Luftdruck

Die Positionierung und Größe der Elemente werden über Koordinaten, Breite, Höhe und gegebenenfalls die Icon-Größe festgelegt.

6.5.2.3 Temperatur (temperature)

Der Temperaturbereich unterscheidet zwischen Innen- und Außenwerten.

Beide Bereiche besitzen eigene Unterelemente für:

- Luftfeuchtigkeit
- Temperatur

Die Luftfeuchtigkeit wird durch einen Icon-Bereich und einen separaten Wertebereich dargestellt.

Zusätzlich ist ein Bereich buttons vorgesehen.

6.5.2.4 Wind und Mond (windmoon)

Dieser Bereich kombiniert meteorologische Winddaten mit Informationen zur Mondphase.

Element	Funktion
icon_direction	Windrichtung
speed	Windgeschwindigkeit
icon_moon	Mond-Icon
moonrise	Mondaufgang
moonset	Monduntergang
phase	Mondphase

Die Darstellung der Symbole und Texte wird über die jeweilige Position, Größe und Textformatierung gesteuert.

6.5.2.5 Kalender (calendar)

Der Kalenderbereich zeigt anstehende Termine an.
Die Darstellung wird über die Höhe eines Termineintrags und den vertikalen Abstand zwischen Einträgen gesteuert.
Jeder Termin besteht aus drei Anzeigeelementen:

Element	Inhalt
day	Tagesangabe
time	Uhrzeit
description	Terminbeschreibung

Die Anzahl der dargestellten Termine und die Auswahl der anzuzeigenden Einträge werden durch diese Layout-Konfiguration nicht ausdrücklich festgelegt.

6.5.3 Gemeinsame Parameter der Anzeigeelemente

Die Anzeigeelemente verwenden abhängig von ihrer Funktion verschiedene Layoutparameter.

Parameter	Bedeutung
x	Horizontale Position
y	Vertikale Position
width	Breite
height	Höhe
font	Referenz auf eine Schriftdefinition
anchor	Ausrichtung beziehungsweise Verankerung
size	Größe eines Icons
visible	Sichtbarkeit eines Elements, sofern vom Renderer ausgewertet

Die Koordinaten beziehen sich auf den jeweiligen übergeordneten Anzeigebereich, sofern die Implementierung die Positionen relativ zum Elternbereich interpretiert.

Die Bedeutung und Auswertung der einzelnen Parameter ist letztlich durch den Renderer bestimmt.

6.5.4 Hierarchisches Konfigurationsprinzip

Die Konfiguration folgt einem hierarchischen Prinzip.

Ein übergeordneter Bereich definiert seine Position und Abmessungen. Darunterliegende Elemente beschreiben die Inhalte dieses Bereichs.

Beispielsweise:

```
Display
└── Elements
    ├── Wordclock
    │   └── Minute Dots
    │
    └── Information
        ├── Date
        ├── Weather
        ├── Temperature
        │   ├── Inside
        │   └── Outside
        ├── Windmoon
        └── Calendar
            └── Event
                ├── Day
                ├── Time
                └── Description
```

Dieses Prinzip ermöglicht eine strukturierte Beschreibung der Benutzeroberfläche und erleichtert die Anpassung des Layouts.

6.5.5 10. Abgrenzung zwischen Layout und Programmlogik

Die Konfiguration beschreibt primär die Darstellung und Anordnung der Anzeigeelemente.
Die eigentliche Datenbeschaffung, Berechnung und Verarbeitung erfolgt in den zugehörigen Programmmodulen.

Beispiele:

- Das Wettermodul beschafft Wetterdaten.
- Das Kalendermodul verarbeitet Kalendertermine.
- Der Renderer setzt die konfigurierten Layoutparameter in eine konkrete Bildschirmdarstellung um.

Nicht alle Eigenschaften der Benutzeroberfläche werden ausschließlich durch die JSON-Datei bestimmt. Das tatsächliche Verhalten hängt auch von der Implementierung des Renderers und der jeweiligen Module ab.

7 System JSON Konfiguration

Die Datei system.json enthält zentrale Konfigurationsparameter für den Betrieb von QLock. Sie legt die Anzeigeeinstellungen fest und konfiguriert die Datenquellen für Kalender, Wetter und Sensoren.

Die Konfiguration ist im JSON-Format aufgebaut und in vier Hauptbereiche unterteilt:

Bereich	Aufgabe
display_layout	Konfiguration des Displays und des Fensterverhaltens
calendar	Konfiguration des Kalenderabrufs
weather	Konfiguration des Wetterdienstes
sensors	Definition und Aktivierung einzelner Sensoren und Datenquellen

7.1 Display-Konfiguration (display_layout)

Dieser Bereich definiert die grundlegenden Eigenschaften der grafischen Anzeige.

Parameter	Datentyp	Beispielwert	Bedeutung
layout	String	"1280x720"	Festlegung des Displaylayouts
fullscreen	Boolean	false	Aktiviert oder deaktiviert den Vollbildmodus
default_bg	String	"black"	Standard-Hintergrundfarbe
cursor	Boolean	false	Legt fest, ob der Mauszeiger angezeigt wird

7.2 Kalender (calendar)

Dieser Bereich konfiguriert den Abruf von Kalenderdaten im iCalendar-Format.

Parameter	Datentyp	Beispielwert	Bedeutung
url	String	Outlook-ICS-URL	Adresse der Kalenderdatei
timeout_seconds	Integer	10	Vorgesehene Zeitgrenze für den Abruf in Sekunden
interval_seconds	Integer	60	Konfiguriertes Abrufintervall in Sekunden

Die Kalenderkonfiguration ermöglicht den regelmäßigen Abruf von Terminen über eine URL.

Die angegebene URL enthält mit CID einen Platzhalter für die Auslieferungskonfiguration.

7.3 Wetter (weather)

Dieser Bereich enthält die Konfigurationsparameter für den Wetterdienst WeatherAPI und den Abruf der Wetter-Icons.

Parameter	Datentyp	Beispielwert	Bedeutung
interval_seconds	Integer	300	Konfiguriertes Aktualisierungsintervall der Wetterdaten
timeout_seconds	Integer	10	Vorgesehene Zeitgrenze für Netzwerkanfragen in Sekunden
api_key	String	"API"	API-Schlüssel des Wetterdienstes
location	String	"49.58,8.64"	Geografische Koordinaten des Wetterstandorts
icon_download_size	String	"64x64"	Gewünschte Größe der Wetter-Icons
base_url	String	WeatherAPI-CDN-Adresse	Basisadresse für den Abruf der Wetter-Icons

Der Wert API ist ein Platzhalter. Bei der Inbetriebnahme muss er durch einen gültigen API-Schlüssel ersetzt werden.

Die Koordinaten legen den konfigurierten Wetterstandort fest.

7.4 Sensoren (sensors)

Der Bereich sensors definiert die verfügbaren Sensoren und externen Datenquellen. Jeder Sensor wird unter einem eigenen Namen konfiguriert. Für intern und extern darf jeweils nur ein Sensor „active“ sein, aber es können beliebig viele Sensoren eingetragen sein.

Die vorliegende Konfiguration enthält die Einträge WeatherAPI und Wohnzimmer.

7.4.1 Parameter eines Sensoreintrags

Parameter	Datentyp	Beispielwert	Bedeutung
ref	String	URL oder Beschreibung	Referenz auf die Datenquelle beziehungsweise zusätzliche Information
type	String	"url" oder "weatherapi"	Typ der Datenquelle
display	String	"intern" oder "extern"	Zuordnung zur Anzeige
interval_seconds	Integer	60	Konfiguriertes Abfrageintervall in Sekunden
active	Integer	0	Aktivierungskennzeichen des Eintrags

7.4.1.1 Konfigurierter Sensor WeatherAPI (default)

Parameter	Wert
ref	"this is default external temperature"
type	"weatherapi"
display	"extern"
interval_seconds	60
active	1

Dieser Eintrag beschreibt eine externe Temperaturdatenquelle auf Basis von WeatherAPI.

7.4.1.2 Konfigurierter Sensor Wohnzimmer

Parameter	Wert
type	"url"
ref	"http://balkon.local/output"
display	"intern"
interval_seconds	60
active	0

Dieser Eintrag beschreibt eine Datenquelle, deren Werte über eine HTTP-Adresse abgerufen werden sollen. Die Datenquelle ist der Anzeige intern zugeordnet.

7.5 Zusammenfassung der Konfigurationsstruktur

Die hierarchische Struktur lässt sich folgendermaßen darstellen:

```
system.json
│
├── display_layout
│   ├── layout
│   ├── fullscreen
│   ├── default_bg
│   └── cursor
│
├── calendar
│   ├── url
│   ├── timeout_seconds
│   └── interval_seconds
│
├── weather
│   ├── interval_seconds
│   ├── timeout_seconds
│   ├── api_key
│   ├── location
│   ├── icon_download_size
│   └── base_url
│
└── sensors
    ├── WeatherAPI
    │   ├── ref
    │   ├── type
    │   ├── display
    │   ├── interval_seconds
    │   └── active
    │
    └── Wohnzimmer
        ├── type
        ├── ref
        ├── display
        ├── interval_seconds
        └── active
```

Grundprinzip

Die Datei trennt Konfigurationsdaten von der Programmlogik.

Dadurch können Einstellungen wie Displaymodus, Kalenderadresse, Wetterstandort und Sensorquellen angepasst werden, ohne die zugehörigen Python-Module direkt verändern zu müssen.

Die tatsächliche Wirkung einzelner Parameter hängt davon ab, ob und wie das jeweilige Programmmodul diese Werte ausliest und verarbeitet.

8 Framework-Module

8.1 framework/logger.py – Logger

Der Logger vereinheitlicht die Protokollierung aller Module. Jedes Modul kann über logger.get("KÜRZEL") einen eigenen ModuleLogger beziehen. Dadurch trägt jede Meldung automatisch den Modulkontext, ohne dass das Kürzel bei jedem Aufruf wiederholt werden muss.

Methode / Eigenschaft	Bedeutung
debug(message)	Detaillierte Diagnosemeldungen für Entwicklung und Fehlersuche.
info(message)	Normale Statusmeldungen, etwa erfolgreicher Start oder Datenaktualisierung.
warning(message)	Ungewöhnliche Situation; der Betrieb kann möglicherweise fortgesetzt werden.
error(message)	Fehler bei einer Verarbeitung.
critical(message)	Schwerwiegender Fehler, der besondere Aufmerksamkeit erfordert.
exception(message)	Protokolliert die Meldung und den aktuellen Stacktrace; innerhalb eines except-Blocks verwenden.
get(module)	Liefert einen modulspezifischen Logger-Kontext.
level	Mindest-Schweregrad; z. B. INFO unterdrückt DEBUG-Meldungen.
maxsize / retries	Grenze für die Logdateigröße und Anzahl der Schreibversuche.

Der MCP erstellt den Logger mit dem Pfad ./logs/application.log. Das Verzeichnis muss beschreibbar sein; die konkrete Initialisierung kann es je nach Implementierung selbst anlegen. Der Logger unterstützt Konsolen- und Dateiausgabe sowie erneute Schreibversuche.

Bekannte Einschränkung: Die aktuelle Größenbehandlung leert die Logdatei beim Erreichen der Grenze, statt ältere Protokolle in nummerierte Archivdateien umzubenennen. Wichtige Fehlerprotokolle daher bei Bedarf vor einer Rotation sichern.

framework/storage.py – Storage

Storage trennt dauerhafte Konfiguration von flüchtigen Laufzeitdaten. Die Systemkonfiguration wird in system.json verwaltet; Laufzeitinformationen werden im Speicher gehalten. Zusätzlich kann runtime.json als Diagnose-Snapshot geschrieben werden. Ein Snapshot ist nicht automatisch gleichbedeutend mit einer vollständig wiederherstellbaren Sitzung.

Datenbereich	Zweck	Hinweise
system	Dauerhafte Einstellungen, die über Neustarts hinweg gelten sollen	Änderungen bewusst speichern; gültiges JSON ist erforderlich.
runtime	Aktueller Zustand der Module während der Laufzeit	Primär für den laufenden Prozess; nicht als dauerhafte Wahrheit behandeln.
runtime.json	Periodischer Snapshot des Laufzeitzustands	Hilfreich für Diagnose; Snapshot-Intervall laut Framework standardmäßig 300 Sekunden.

Storage bietet unter anderem begin(), service(), stop(), save_system() und save_runtime(). Die Lade-/Speicherfunktionen zentralisieren Dateizugriffe und vermeiden, dass jedes Fachmodul seine eigenen JSON-Dateien unabhängig verwalten muss.

8.2 framework/timers.py – Timers

Timers stellt Hilfsfunktionen für die lokale Systemzeit und Zeitabstände bereit. Es ist kein Scheduler und startet keine eigenen Threads. Fachmodule entscheiden selbst, wann eine Aktualisierung fällig ist.

Funktion	Verwendung
now()	Aktuelle lokale Systemzeit als datetime.
elapsed(start, seconds)	Prüft, ob seit start mindestens die angegebene Zeit vergangen ist; bei start=None wird True geliefert.
seconds_since(start)	Verstrichene Sekunden seit einem Zeitpunkt.
seconds_until(target)	Zeit bis zu einem Zielzeitpunkt.
minute(now)	Minute der angegebenen bzw. aktuellen Zeit.
minute_id(now)	Kennung der aktuellen Minute, um Minutenwechsel zu vergleichen.
minute_changed(last_minute, now)	Erkennt einen Minutenwechsel.
next_minute(now)	Berechnet den Beginn der nächsten Minute.
seconds_to_next_minute(now)	Sekunden bis zur nächsten vollen Minute.
second(), time_text(), date_text(), weekday_text(), long_date_text()	Zeitbestandteile und formatierte Anzeigezeichenfolgen.

Zeitbasis: Die Funktionen verwenden datetime.now() und damit die lokale Systemzeit des Raspberry Pi. Zeitzone und Sommer-/Winterzeit hängen von der Betriebssystemkonfiguration ab. Die datetime-Werte enthalten keine explizite Zeitzoneninformation.

9 Anwendungs-Module

9.1 modules/scheduler.py – Zeitsteuerung

Der Scheduler bündelt zeitabhängige Aktualisierungen und Erinnerungslogik. Er stellt einen eigenen Lebenszyklus bereit und kann in service() prüfen, ob zeitbezogene Arbeit ansteht. update() verarbeitet den aktuellen Zeitpunkt; update_reminder() kümmert sich um Erinnerungen. Der Scheduler ist von den reinen Zeit-Hilfsfunktionen in Timers zu unterscheiden: Timers berechnet, Scheduler entscheidet fachlich, was bei einem Zeitpunkt geschehen soll.

9.2 modules/wordclock.py – Wortuhr

WordClock liest die Wortuhr-Konfiguration aus dem Konfigurationsbereich, darunter Wortmatrix, Wörter, Sequenzen und Stundenbezeichnungen. update() aktualisiert den Zeitstatus; get_words() bestimmt die anzuzeigenden Wörter; create_matrix() überführt die Auswahl in eine Matrixdarstellung. So bleibt die sprachliche Zeitlogik von der konkreten Pixel-/Canvas-Zeichnung getrennt.

9.3 modules/calendar.py – Kalender

Calendar lädt Kalenderdaten im ICS-Format über HTTP und verarbeitet sie mit icalendar sowie recurring_ical_events. Die Wiederholungsbibliothek ist wichtig, weil wiederkehrende Termine in einer ICS-Datei nicht zwingend als einzelne Ereignisse enthalten sind. check_calendar() prüft den Aktualisierungsbedarf; download_calendar() lädt und verarbeitet die Quelle.

Für den Betrieb müssen Kalenderquelle, Netzwerkzugriff und die benötigten Python-Pakete korrekt eingerichtet sein. Fehler beim Abruf sollten in den Logmeldungen von einem tatsächlich leeren Kalender unterschieden werden.

9.4 modules/sensors.py – Sensorverwaltung

Das Modul enthält die Klasse Sensor als einzelne Datenquelle und die Klasse Sensors als zentrale Verwaltung. Die Sensor-Konfiguration legt unter anderem Typ, Referenz, Anzeigeverhalten und Aktualisierungsintervall fest. get_data() fragt die jeweilige Quelle ab; die Verwaltung aktualisiert Werte und stellt Namen für Innen- und Außensensoren bereit.

Der Quellcode unterstützt unter anderem einen weatherapi-Datenpfad. Sensorwerte sollten auf Gültigkeit und Aktualität geprüft werden, bevor sie angezeigt oder weiterverarbeitet werden. Bei Netzsensoren sind API-Schlüssel, URL und Erreichbarkeit installationsabhängige Voraussetzungen.

9.5 modules/weather.py – Wetter und Astronomie

Weather ruft Wetter-/Vorhersagedaten ab und bereitet sie für andere Module auf. Dazu gehören Wetterzustand, Wind, astronomische Angaben sowie die Erzeugung bzw. Auswahl passender Wetter-, Wind- und Mond-Icons. Die Daten werden als Laufzeitzustand bereitgestellt; die Darstellung selbst erfolgt in den Renderer-Modulen.

Zu den Aufgaben zählen make_weather_icon(), make_wind_icon(), make_moon_icon(), prepare_weather(), moon_phase(), Zeitnormalisierung und update(). Netzwerkfehler, fehlende API-Daten und fehlende Icon-Dateien sind typische Fehlerquellen, die bei der Diagnose getrennt betrachtet werden sollten.

9.6 modules/renderer.py – Fenster und Canvas

Renderer verwaltet die Tkinter-Oberfläche: Konfiguration laden, Layout auswählen/validieren, Fenster und Canvas erzeugen, Aktualisierung anstoßen und auf Maus-/Vollbildereignisse reagieren. Die Layout-Konfiguration kommt aus system.json und verweist auf ein Layout. Für den Vollbildbetrieb müssen Bildschirmauflösung und Layoutkoordinaten zusammenpassen.

9.7 modules/renderer_display.py – Display-Koordination

RendererDisplay verbindet die eigentliche Wortuhrdarstellung mit den Informationsbereichen. Es steuert deren begin(), render() und stop() und stellt gemeinsame Hilfsinformationen wie Elemente, Tages-/Nachtstatus, Farben und Icon-Namen bereit.

9.8 modules/renderer_display_wordclock.py – Wortuhr zeichnen

Dieses Modul zeichnet die Wortmatrix und die Minutenpunkte. Layoutparameter wie x, y, width, height, Schrift, Buchstabenabstände und minute_dots stammen aus dem Layout. Die Bilddateien für Minutenpunkte werden anhand von Größenpfaden unter icons/images gesucht. Bei Darstellungsproblemen zuerst Layoutwerte, Schriftkonfiguration und die erwarteten Bilddateien kontrollieren.

9.9 modules/renderer_display_information.py – Informationsanzeige

Dieser Koordinator verwaltet die Teilbereiche für Datum, Kalender, Temperatur, Wetter sowie Wind/Mond. Dadurch können Informationsbereiche unabhängig entwickelt werden, ohne die Fensterverwaltung oder Wortuhrlogik ändern zu müssen.

9.10 Unterbereiche der Informationsanzeige

Datei	Zuständigkeit
renderer_display_information_date.py	Datum und Zeittexte sowie deren Platzierung.
renderer_display_information_calendar.py	Darstellung der nächsten Kalendertermine.
renderer_display_information_temperature.py	Innen-/Außentemperatur, Luftfeuchtigkeit, Icons und Tag-/Nachtfarben.
renderer_display_information_temperature_buttons.py	Schaltflächen zur Auswahl des anzuzeigenden Sensors.
renderer_display_information_weather.py	Wettertexte und Wetter-Icon.
renderer_display_information_wind_moon.py	Wind- und Monddarstellung einschließlich Symbolen.

Alle Teilbereiche verwenden gemeinsame Layout-/Elementinformationen. Ein Bereich kann daher korrekt Daten besitzen und trotzdem unsichtbar bleiben, wenn sein Layoutname, seine Geometrie, Farben, Schrift oder Bilddateien nicht passen.

10 Konfiguration, Datenquellen und Laufzeitdaten

10.1 Datenquellen konfigurieren

10.1.1 WeatherAPI

https://www.weatherapi.com/ stellt bereits einen Teil der Wetterdaten zur Verfügung, die standardmäßig genutzt werden, wie Temperatur, Mond und Sonne und Wetter-Icons. Es wird ein Schlüssel für die Abfrage benötigt (Get Free API Key), hierzu ist eine E-Mail-Adresse erforderlich. Anschließend den API-Key kopieren.

10.1.2 Microsoft Calendar vorbereiten

- Im Event-Bereich (Bereich 5) in der Informationsanzeige werden Daten eines Microsoft-Kalenders angezeigt. Entweder hat man bereits ein Konto, oder man kann dieses kostenfrei anlegen. Für die Verarbeitung der Kalenderdaten in QLock ist der ICS-Link erforderlich.
- Anmelden im Microsoft Konto in einem Browser, Kalender auswählen
- Im Menü “Anzeigen” und “Kalendereinstellungen” wählen
- dann auf “Geteilte Kalender”
- auf der rechten Seite erscheint dann der HTML- und ICS-Link des Kalenders
- den ICS-Link kopieren

10.2 system.json konfigurieren

Storage stellt die dauerhafte Systemkonfiguration unter ./configuration/* bereit. Der Renderer erwartet darin insbesondere display_layout und darin den Namen des zu verwendenden Layouts. Die WordClock erwartet eine Wortuhr-Konfiguration mit wordmatrix, words, sequences und hours. Die Sensor-, Kalender- und Wettermodule benötigen darüber hinaus ihre jeweiligen Einstellungen.

10.3 Layout und Icons

Renderer und Display-Teilmodule verwenden Layoutdaten mit Elementnamen und Geometrie. Die Wortuhr greift außerdem auf Icons/Bilder unter icons bzw. ./icons/images zu. Dateinamen und Auflösungsvarianten müssen exakt zu den im Quellcode zusammengesetzten Pfaden passen.

10.4 Laufzeitdateien und Logs

Datei / Pfad	Funktion	Prüfen bei
system.json	Dauerhafte Konfiguration	Startfehler, fehlende Schlüssel, unerwartete Darstellung
runtime.json	Diagnose-Snapshot des Laufzeitzustands	Veraltete oder unerwartete Laufzeitwerte
./logs/application.log	Zentrale Protokolldatei	Start-, Netzwerk-, Daten- und Darstellungsfehler
configuration/...	Konfigurationsdateien für Layout und Wortuhr	Datei nicht gefunden oder Layout/Matrix unvollständig
icons/...	Wetter-, Wind-, Mond- und Minutenbilder	Fehlende Symbole oder leere Bildbereiche

11 Installation und Inbetriebnahme

Das ZIP-Paket herunterladen und entpacken. Die Struktur der Verzeichnisse muss erhalten bleiben.

11.1 Vorbereitung Entwicklungsumgebung Windows

Aus dem Microsoft Store herunterladen

- Python mit Thonny
  https://apps.microsoft.com/detail/9PNRBTZXMB4Z?hl=neutral&gl=DE&ocid=pdpshare
- Arduino IDE
  https://apps.microsoft.com/detail/XPDDTBJ80F8PC9?hl=de-DE&gl=DE&ocid=pdpshare
- Optional: Ein Editor für .json und sonstige Programmierung. Der Editor Notepad++
  https://notepad-plus-plus.org/

dann weiter mit Vorbereitung Datenquellen

11.2 Vorbereitung Entwicklungsumgebung Raspberry

Bei der Neu-Installation des Raspberry mit Raspberry Pi Imager direkt WLAN-Zugang und Remote-Access konfigurieren. Nach dem ersten Start auf Updates prüfen und installieren. Die Python-Umgebung ist bereits vorinstalliert. Schreibrechte für die Verzeichnisse prüfen. Dann weiter mit Vorbereitung Datenquellen

11.3 Vorbereitung Datenquellen

11.3.1 WeatherAPI

https://www.weatherapi.com/ stellt bereits einen Teil der Wetterdaten zur Verfügung, die standardmäßig genutzt werden, wie Temperatur, Mond und Sonne und Wetter-Icons. Es wird ein Schlüssel für die Abfrage benötigt (Get Free API Key), hierzu ist eine E-Mail-Adresse erforderlich. Anschließend den API-Key kopieren. Dieser ersetzt “**API**” in der system.json (weather->api_key = “**API**”).
Über https://www.google.com/maps den gewünschten Standort suchen und in die Karte klicken. Es erscheinen dann die geografischen Koordinaten (z.B. 49.5752724,8.596522). Diese Koordinaten (z.B. 49.57,8.59) ersetzen “**LOCATION**” in der system.json (weather->location = “**LOCATION**”.

11.3.2 Kalender

QLock liest Kalenderdaten aus einer URL, die eine Datei im iCalendar-Format (ICS) bereitstellt. Dadurch ist die Kalenderverarbeitung grundsätzlich anbieterunabhängig: Neben Microsoft Outlook kann beispielsweise auch Google Kalender ohne Änderung am QLock-Code verwendet werden, sofern eine geeignete ICS-Adresse verfügbar ist. Die Aktualisierung der Termine hängt auch davon ab, wann der Kalenderanbieter Änderungen im Feed bereitstellt.
Sicherheit: Eine geheime/private ICS-Adresse ist wie ein Zugangsschlüssel zu behandeln. Sie darf nicht in öffentliche Dokumentationen, Screenshots oder öffentlich zugängliche Repositories übernommen werden. Für die Auslieferung ist ein neutraler Platzhalter zu verwenden.

11.3.2.1 Microsoft Calendar

- Im Event-Bereich (Bereich 5) in der Informationsanzeige werden Daten eines Microsoft-Kalenders angezeigt. Entweder hat man bereits ein Konto, oder man kann dieses kostenfrei anlegen. Für die Verarbeitung der Kalenderdaten in QLock ist der ICS-Link erforderlich.
- Mit einem Browser bei Microsoft https://outlook.live.com/calendar anmelden, Kalender auswählen
- Im Menü “Anzeigen” und “Kalendereinstellungen” wählen
- dann auf “Geteilte Kalender”
- auf der rechten Seite erscheint dann der HTML- und ICS-Link des Kalenders
- Den kopierten ICS-Link als Wert für calendar.url in system.json eintragen. Der Platzhalter CID in der ausgelieferten Beispieladresse wird dadurch vollständig ersetzt.

11.3.2.2 Google Calendar

- im Google Kalender auf Einstellungen
- dann “Einstellung für meinen Kalender”
- auf den entsprechenden Kalender klicken
- auf der rechten Seite dann unter “Kalender integrieren”
- den ICS Link “Privatadresse im iCal-Format” kopieren
- Den kopierten ICS-Link als Wert für calendar.url in system.json eintragen. Der Platzhalter CID in der ausgelieferten Beispieladresse wird dadurch vollständig ersetzt.

11.3.3 Sensoren

Die system.json kann Sensoren vorkonfigurieren. Ob ein Sensor tatsächlich aktiviert ist, ergibt sich aus seinem active-Wert und der Auswertung im Sensor-Modul. Vor der Inbetriebnahme sollte geprüft werden, welcher Sensor für intern beziehungsweise extern verwendet werden soll. Ist eine Datenquelle nicht erreichbar oder deaktiviert, kann die Anzeige ohne entsprechende Werte bleiben. QLock erwartet von der konfigurierten URL eine JSON-Nachricht im Format

11.3.4 Display Layout

Es ist ein gültiges Display-Format in der system.json (display_layout->layout) anzugeben. Derzeit ist 1280x720 voreingestellt.
Neue Display-Formate können einfach neu angelegt werden. Einfach ein bestehendes Layout aus ./configuration/display kopieren und die Parameter der Anzeigen entsprechend anpassen. Im Verzeichnis ./icons/images muss ebenfalls ein entsprechendes Format angelegt sein.

11.4 Vorbereitung Thonny (Python)

11.4.1 Virtuelle Umgebung erstellen

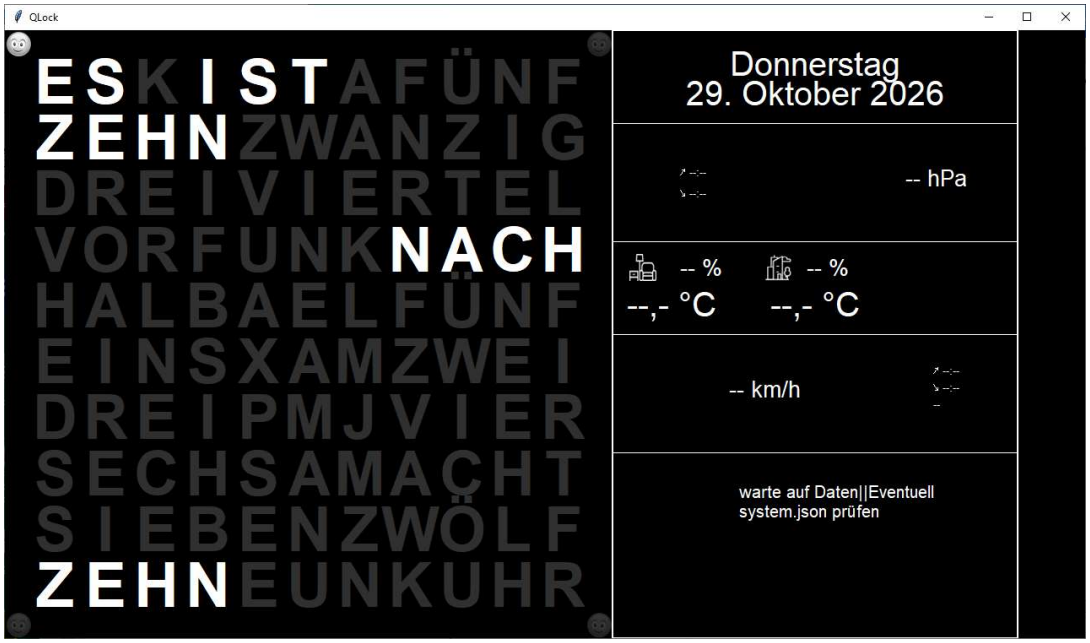
- in Thonny das Menü Werkzeuge->Optionen aufrufen
- im Reiter Interpreter unten rechts „new virtual environment“ klicken
- anschließend ein leeres Verzeichnis (./_Environment ist dafür vorgesehen) auswählen und OK

11.4.2 Bibliotheken installieren

- Im Verzeichnis ./_Installation das Script install.py in Thonny öffnen und mit F5 ausführen. Vorher in Thonny den Interpreter auswählen, in dem QLock betrieben werden soll. Das Script aktualisiert zuerst pip und installiert danach die in requirements.txt aufgeführten externen Python-Pakete in genau dieser Python-Umgebung.
- Unter Raspberry Pi OS kann tkinter zusätzlich als Betriebssystempaket erforderlich sein (zum Beispiel python3-tk). Python-Pakete, die über pip installiert werden, und Betriebssystempakete sind getrennt zu behandeln.
- das Script schließen

11.5 Erster Start QLOCK 2.0

Die Datei mcp.py in Thonny laden und mit F5 starten. Die Wortuhr funktioniert auch ohne korrekt konfigurierte Sensoren; Datenbereiche können dann ohne Werte erscheinen.



Im Direktfenster von Thonny kann nachvollzogen werden, dass die Datenquellen noch nicht vollständig konfiguriert sind. Das folgende Protokoll ist ein Beispiel aus einem Testlauf mit Platzhalter- beziehungsweise ungültigen Zugangsdaten. Die Meldungen zu Kalender und Wetter sind in diesem Kontext erwartete Konfigurationsfehler und kein Beleg für einen generellen Fehler der QLock-Anwendung:

```
----------------------------------BOOT WORDCLOCK 2.0----------------------------------
2026-10-29 22:08:48.732 [INFO   ] [LOG  : 178 begin()] Framework Logger erfolgreich gestartet
2026-10-29 22:08:48.732 [INFO   ] [TIM  :  64 begin()] Framework Timers erfolgreich gestartet
2026-10-29 22:08:48.734 [INFO   ] [STO  : 136 begin()] Framework Storage erfolgreich gestartet
2026-10-29 22:08:48.734 [ERROR  ] [CAL  : 172 begin()] Calendar: Konfigurationsfehler: url muss mit http:// oder https:// beginnen
2026-10-29 22:08:48.734 [INFO   ] [WTH  : 261 begin()] Modul Weather gestartet
2026-10-29 22:08:48.802 [ERROR  ] [WTH  : 682 update()] WeatherAPI: HTTP-Fehler 401; Wetterdaten konnten nicht geladen werden.
2026-10-29 22:08:48.803 [INFO   ] [SDL  : 150 begin()] Modul Scheduler gestartet
2026-10-29 22:08:48.887 [INFO   ] [R_DSP_WRD :  90 begin()] Sektion WordClock gestartet
2026-10-29 22:08:48.887 [INFO   ] [R_DSP_INF_DAT : 105 begin()] Sektion Date gestartet
2026-10-29 22:08:48.888 [INFO   ] [R_DSP_INF_CAL :  78 begin()] Sektion Calendar gestartet
2026-10-29 22:08:48.888 [INFO   ] [R_DSP_INF_TMP :  78 begin()] Sektion Temperaturanzeige gestartet
2026-10-29 22:08:48.889 [INFO   ] [R_DSP_INF_WTH : 105 begin()] Sektion Weather gestartet
2026-10-29 22:08:48.889 [INFO   ] [R_DSP_INF_WMO : 126 begin()] Sektion WindMoon gestartet
2026-10-29 22:08:48.890 [INFO   ] [R_DSP_INF :  80 begin()] Sektion Information gestartet
2026-10-29 22:08:48.890 [INFO   ] [R_DSP :  73 begin()] RendererDisplay gestartet
2026-10-29 22:08:48.891 [INFO   ] [REN  :  86 begin()] Renderer gestartet: 1280x720
2026-10-29 22:08:48.892 [INFO   ] [CLK  : 175 begin()] Modul WordClock gestartet
2026-10-29 22:08:48.892 [INFO   ] [SNS  : 402 begin()] Modul Sensors gestartet
2026-10-29 22:08:48.893 [INFO   ] [SNS  : 507 update()] Sensoren aktualisiert
----------------------------------STARTUP COMPLETED----------------------------------
```

----------VIEL SPASS----------

12 Betrieb, Fehlerdiagnose und Wartung

Symptom	Mögliche Ursache	Vorgehen
Anwendung startet nicht	Fehlendes Paket, falsches Arbeitsverzeichnis, fehlende Konfiguration	Terminalausgabe und application.log prüfen; Dateipfade und Importfehler kontrollieren.
Fenster öffnet sich, Anzeige bleibt leer	Layout fehlt/ungültig, Elemente außerhalb der Fläche, Renderer-Fehler	display_layout, Layoutdatei, Elementnamen und Geometrie prüfen.
Wortmatrix stimmt nicht	Wortmatrix-/Sequenzkonfiguration passt nicht zur erwarteten Logik	wordmatrix, words, sequences und hours vergleichen; WordClock getrennt von Renderer testen.
Wetter fehlt	Netzwerk, API-Schlüssel, API-Limit, Antwortformat oder Icon fehlt	Logmeldungen und Weather-Laufzeitdaten prüfen; API-Aufruf separat testen.
Kalendertermine fehlen	ICS-URL nicht erreichbar, ungültige ICS-Datei, Zeitzonen-/Wiederholungsdaten	URL-Zugriff und Parserfehler prüfen; erfolgreiche leere Antwort von Abruffehler unterscheiden.
Sensorwerte veraltet	Intervall, Netzquelle, ungültige Antwort oder deaktivierter Sensor	Sensor-Konfiguration und Zeitstempel/Aktualisierungsstatus kontrollieren.
Logdatei wird nicht geschrieben	logs-Verzeichnis fehlt oder Schreibrechte fehlen	Arbeitsverzeichnis und Benutzerrechte prüfen.
Zeit/Datum falsch	Systemzeit, Zeitzone oder NTP falsch	Betriebssystemzeit und Zeitzone auf dem Raspberry Pi prüfen.
CPU-Last höher als erwartet	Zu häufige Arbeit in service(), unnötiges Zeichnen, langsame Netzaufrufe	Service-Aufrufe zeitlich messen; Netzwerkzugriffe nicht in jedem Durchlauf wiederholen; nur bei Änderung neu zeichnen.

12.1 Sinnvolle Reihenfolge bei der Fehlersuche

- Zuerst die erste Fehlermeldung nach dem Start suchen – Folgefehler können nur Auswirkungen sein.
- Feststellen, ob der Fehler beim Laden der Konfiguration, beim Abruf von Daten oder beim Zeichnen auftritt.
- Das betroffene Modul anhand des Modulkontexts im Log identifizieren.
- Die kleinste betroffene Einheit prüfen, statt gleichzeitig Konfiguration, Datenquelle und Darstellung zu ändern.
- Nach einer Änderung genau einen kontrollierten Neustart durchführen und das Ergebnis protokollieren.

12.2 Logging richtig einsetzen

DEBUG ist für Diagnose und Entwicklung geeignet, INFO für den normalen Betrieb. WARNING sollte unerwartete, aber nicht unmittelbar fatale Situationen markieren. ERROR und CRITICAL sind für Fehler reserviert. In Ausnahmebehandlungen ist exception() sinnvoll, weil der Stacktrace die Fehlerstelle sichtbar macht.

13 Erweiterung und Änderungsregeln

13.1 Ein neues Informationsfeld ergänzen

- Festlegen, welche Datenquelle den Inhalt liefert und wie der Laufzeitwert benannt ist.
- Das Layout-Element mit eindeutiger Kennung, Geometrie, Schrift und Farben definieren.
- Ein eigenes Renderer-Teilmodul ergänzen oder einen vorhandenen Teilbereich erweitern.
- Das neue Teilmodul im Informationskoordinator erzeugen und in begin()/render()/stop() aufnehmen.
- Fehlende Daten, lange Texte und nicht vorhandene Icons robust behandeln.
- Mit gültigen und fehlenden Daten testen und auf unnötige Dauer-Neuzeichnungen achten.

13.2 Ein neues Datenmodul ergänzen

- Das Modul mit klarer Zuständigkeit erstellen und den MCP injizieren.
- Im Konstruktor nur die Abhängigkeiten verwenden, die bereits registriert sind.
- begin() für Initialisierung, service() für kurze fällige Arbeit und stop() für sauberes Beenden verwenden.
- Zeitintervalle über Timers prüfen, statt in service() zu schlafen oder zu blockieren.
- Konfiguration über Storage beziehen und Diagnosemeldungen mit einem modulspezifischen Logger schreiben.
- Das Modul im MCP in der richtigen Reihenfolge registrieren und Fehler-/Shutdown-Verhalten testen.

13.3 Grundregeln für wartbaren Code

- Ein Modul sollte eine klar erkennbare fachliche Verantwortung haben.
- Keine langen time.sleep()-Wartezeiten in einer zentral zyklisch aufgerufenen service()-Methode.
- Netzwerk- und Dateizugriffe auf sinnvolle Intervalle begrenzen.
- Konfigurationswerte validieren und bei fehlenden Pflichtwerten eine verständliche Fehlermeldung erzeugen.
- Runtime-Daten nicht unkontrolliert als dauerhafte Konfiguration behandeln.
- Fehler nicht stillschweigend verschlucken; den Modulkontext im Log erhalten.
- Bei Änderungen an öffentlichen Datenstrukturen auch alle lesenden Renderer und Module prüfen.

14 Glossar

Begriff	Erklärung
MCP	Zentrale Steuerinstanz, die Module und gemeinsame Dienste verwaltet.
Lebenszyklus	Abfolge aus Initialisierung, laufendem Betrieb und geordnetem Beenden.
service()	Kurzer zyklischer Arbeitsschritt eines Moduls; sollte nicht blockieren.
Runtime	Laufzeitdaten, die sich während der Ausführung verändern.
Snapshot	Zeitpunktbezogene Kopie von Laufzeitdaten zur Diagnose.
ICS	Textbasiertes Kalenderformat, das unter anderem von Kalenderdiensten exportiert wird.
Renderer	Komponente, die Daten grafisch auf dem Bildschirm darstellt.
Layout	Konfiguration von Positionen, Größen, Schriften und Anzeigeelementen.
Stacktrace	Aufrufkette, die zeigt, wo und über welche Funktionen eine Ausnahme entstanden ist.
API	Programmierschnittstelle eines externen Dienstes, etwa einer Wetterdatenquelle.

15 Nächste Änderungen/Neue Version

Begriff	Erklärung
Überprüfung Mapping in z.B. 1280x800.json	Tag- und Monatsmapping gehört eigentlich in system.json, nicht in die Display-Konfiguration
Buttons für Temperatursensoren	Mehrere Innen- und Außensensoren sind anwählbar

15.1 Abschließende Hinweise

Die Dokumentation soll als lebendiges Projekthandbuch verwendet werden. Wenn sich die Konfiguration, ein Modul oder die Datenstruktur ändert, sollte der entsprechende Abschnitt mit angepasst werden. Vor einer größeren Erweiterung empfiehlt sich eine Sicherung von Quellcode, Konfiguration, Icons und Laufzeitdateien.

Für eine spätere Dokumentationsrunde empfiehlt sich, die Konfigurationsbeispiele regelmäßig mit den tatsächlich ausgelieferten Dateien abzugleichen, die Installationsschritte auf die verwendete Raspberry-Pi-OS-Version abzustimmen und nicht öffentliche Zugangsdaten ausschließlich durch Platzhalter darzustellen.