OTCv8 Dev — Dokumentacja

Build • Architektura • Lua/OTUI • Realtime

None

None

Table of contents

1. OTCv8 Dev — Dokument 4
2. Przewodniki 6
   - 2.1 Architektura (sk 6
   - 2.2 Moduły (vBot) — przewo 8
     - 2.2.1 Struktura mod 8
     - 2.2.2 Minimalny moduł (L 8
     - 2.2.3 Rejestrowanie zdarzeń (przykł 8
     - 2.2.4 Konfigura 9
     - 2.2.5 Debug / l 9
     - 2.2.6 Dobre prakt 9
   - 2.3 Realtime (WebSoc 11
     - 2.3.1 Zas 11
     - 2.3.2 Przykład (Node + socket 11
   - 2.4 OTUI — pods 13
     - 2.4.1 Przykład layo 13
     - 2.4.2 Zdarzenia / wiąza 13
     - 2.4.3 Wskazó 13
3. Build 15
   - 3.1 Build — Windows 15
   - 3.2 Build — Linux 16
   - 3.3 Build — Android 17
4. Lua 18
   - 4.1 Lua API (przykł 18
   - 4.2 Lua — style g 20
5. C++ 22
   - 5.1 C++ — prze 22
6. Narzę 24
7. Deweloperzy 26
   - 7.1 Contribu 26
     - 7.1.1 St 26
     - 7.1.2 Commity dokumenta 26
   - 7.2 T 28
     - 7.2.1 Raport bł 28
   - 7.3 Rel 30
     - 7.3.1 Wersjonowanie dokumenta 30

7.4 Secu 32
7.5 Troubleshoo 34
7.5.1 Bu 34
7.5.2 Andr 34
7.5.3 34
7.6 36
7.7 Sło 38
7.8 Roa 40

1. OTCv8 Dev — Dokument

a

c

j

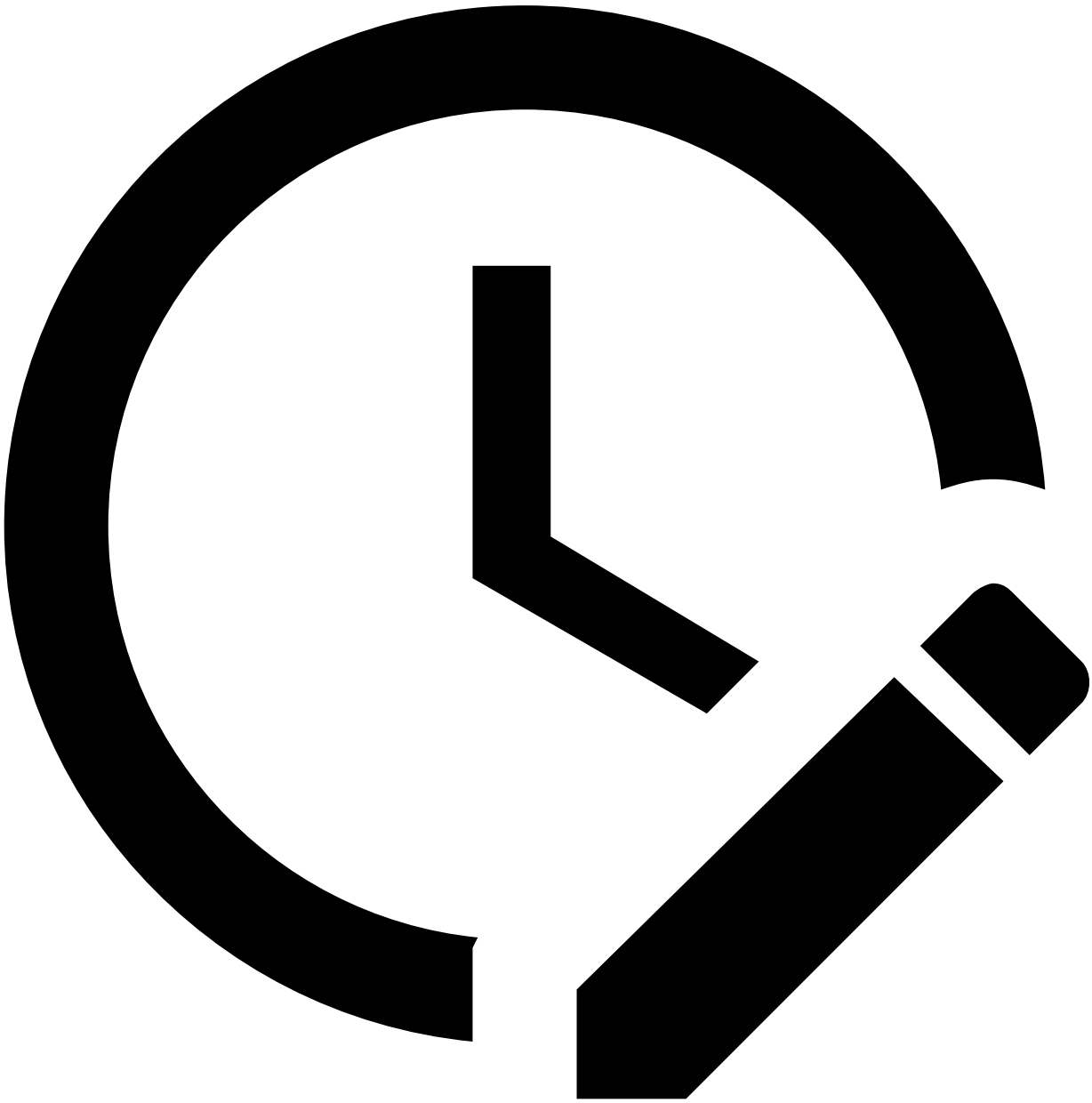
a

Cel: kompletna baza wiedzy (build, architektura, API Lua/C++), z diagramami i przykładami.

- **Szybki start**: sekcja *Build*.
- **Architektura**: diagramy Mermaid + graf modułów.
- **API**: przykłady Lua/C++ (do rozszerzenia).

> Baza wiedzy dla edytora

Twój edytor może konsumować `search/search_index.json` z tej strony (GitHub Pages) jako indeks wiedzy...



3 października 2025

2. Przewodniki

2.1 Architektura (sk

r

ó

t

)

```
graph TD
  Client[OTCv8 Client] -->|Lua| vBot[vBot Modules]
  Client -->|OTUI| UI[UI System]
  Client --> CppCore[C++ Core]
  WS[WebSocket/IPC] --> Client
```

Warstwy

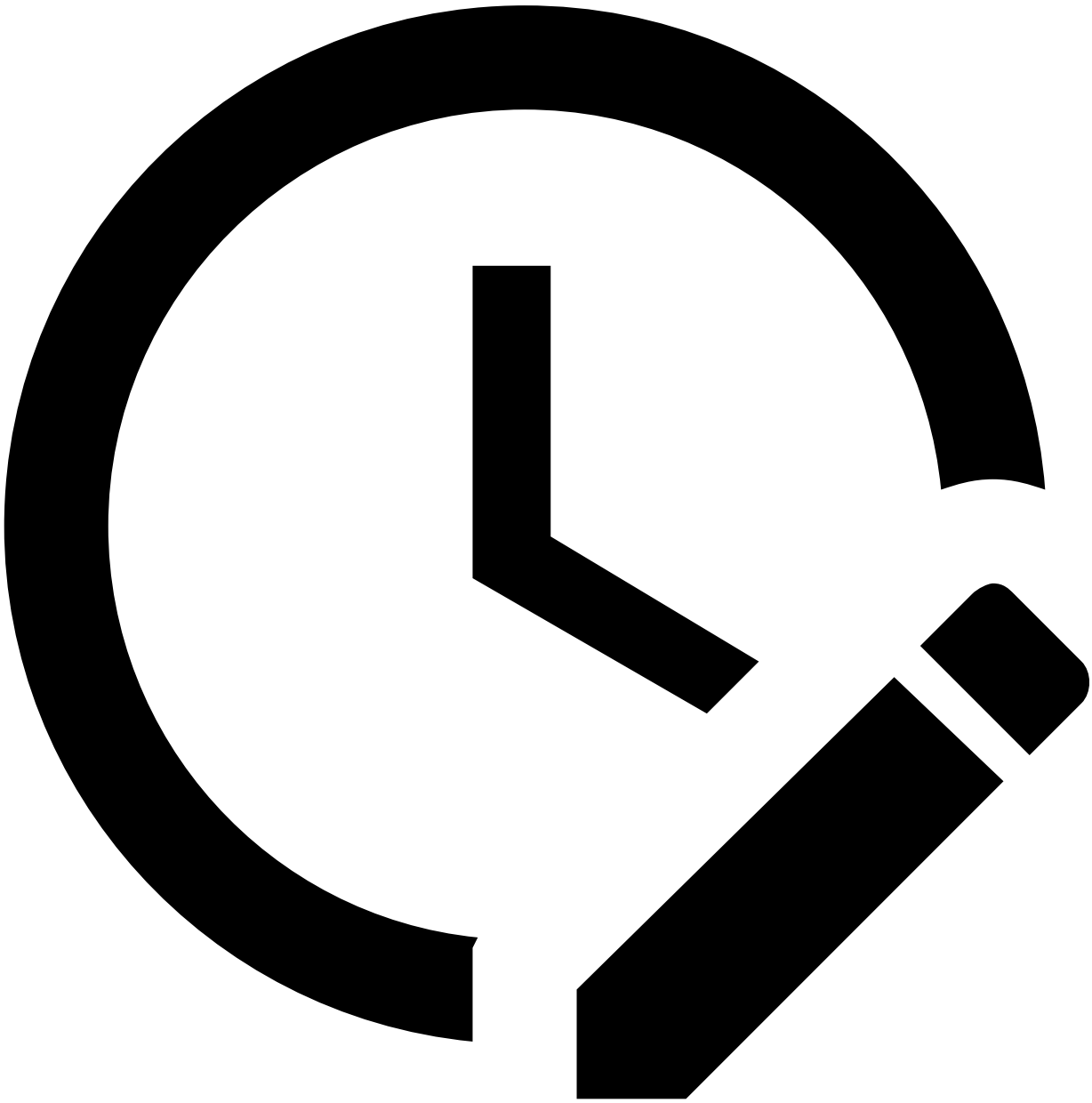
C++ Core – silnik render/UI/IO

Lua – logika modułów (vBot)

OTUI – deklaratywne layouty

Kontrakty

Eventy Lua ⟷ UI, IPC/WS, zasoby



3 października 2025

2.2 Moduły (vBot) — przewo

d

n

i

k

Cel

Jak tworzyć i ładować moduły Lua dla klienta OTCv8.

2.2.1 Struktura mod

u

ł

u

```
modules/
my-module/
init.lua
config.lua
README.md
```

2.2.2 Minimalny moduł (L

u

a

)

```
-- modules/my-module/init.lua
local M = {}

function M.start()
  print("my-module start")
end

function M.stop()
  print("my-module stop")
end

return M
```

2.2.3 Rejestrowanie zdarzeń (przykł

a

d

)

```
onTalk(function(name, level, mode, text)
  if text:find("hello") then print("Hi " .. name) end
end)
```

2.2.4 Konfigura

c

j

a

- `config.lua` – wartości domyślne (np. hotkeye, progi).
- Pliki konfiguracyjne użytkownika trzymaj oddzielnie.

2.2.5 Debug / l

o

g

i

- Pisz do konsoli lub pliku `logs/my-module.log`.
- Dodaj flagę `DEBUG=true` i warunkowe logowanie.

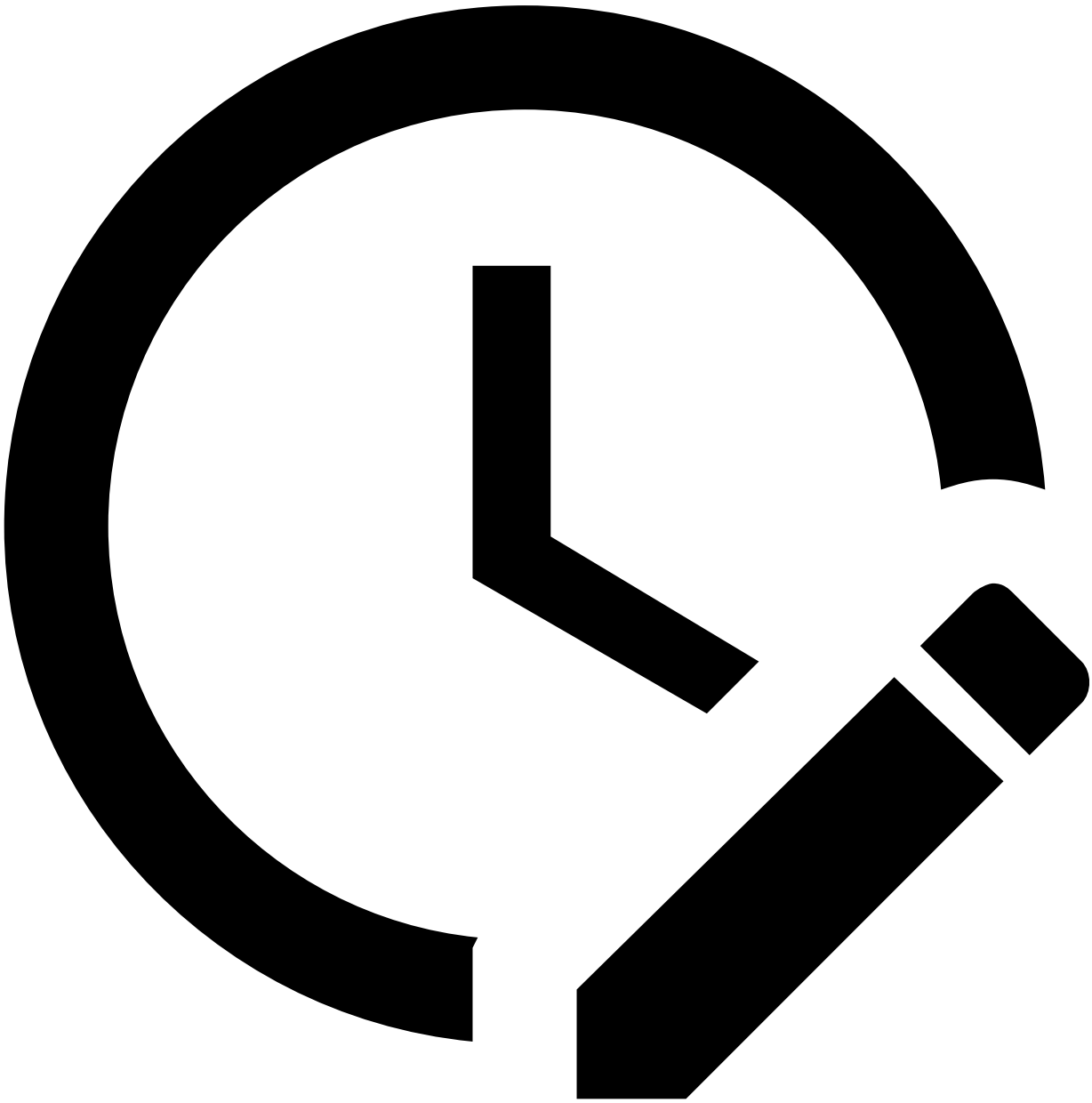
2.2.6 Dobre prakt

y

k

i

- Nazwy przestrzeni modułu (`my_module.*`).
- Brak efektów ubocznych przy `require`.
- Komendy eksportuj jawnie (np. `M.start`, `M.stop`).



3 października 2025

2.3 Realtime (WebSoc

k

e

t

)

```
sequenceDiagram
  participant UI as Dashboard (SPA)
  participant WS as WebSocket (wss)
  participant S as Server (Node)

  UI->>WS: handshake (JWT / token)
  WS->>S: connect
  S-->>UI: events: metrics, logs, char_info
  UI->>S: cmd: START/STOP, settings
```

2.3.1 Zas

a

d

y

- **WSS** + origin allowlist + rate-limit.
- Autoryzacja w handshake (JWT / session).
- Walidacja schematów wiadomości.

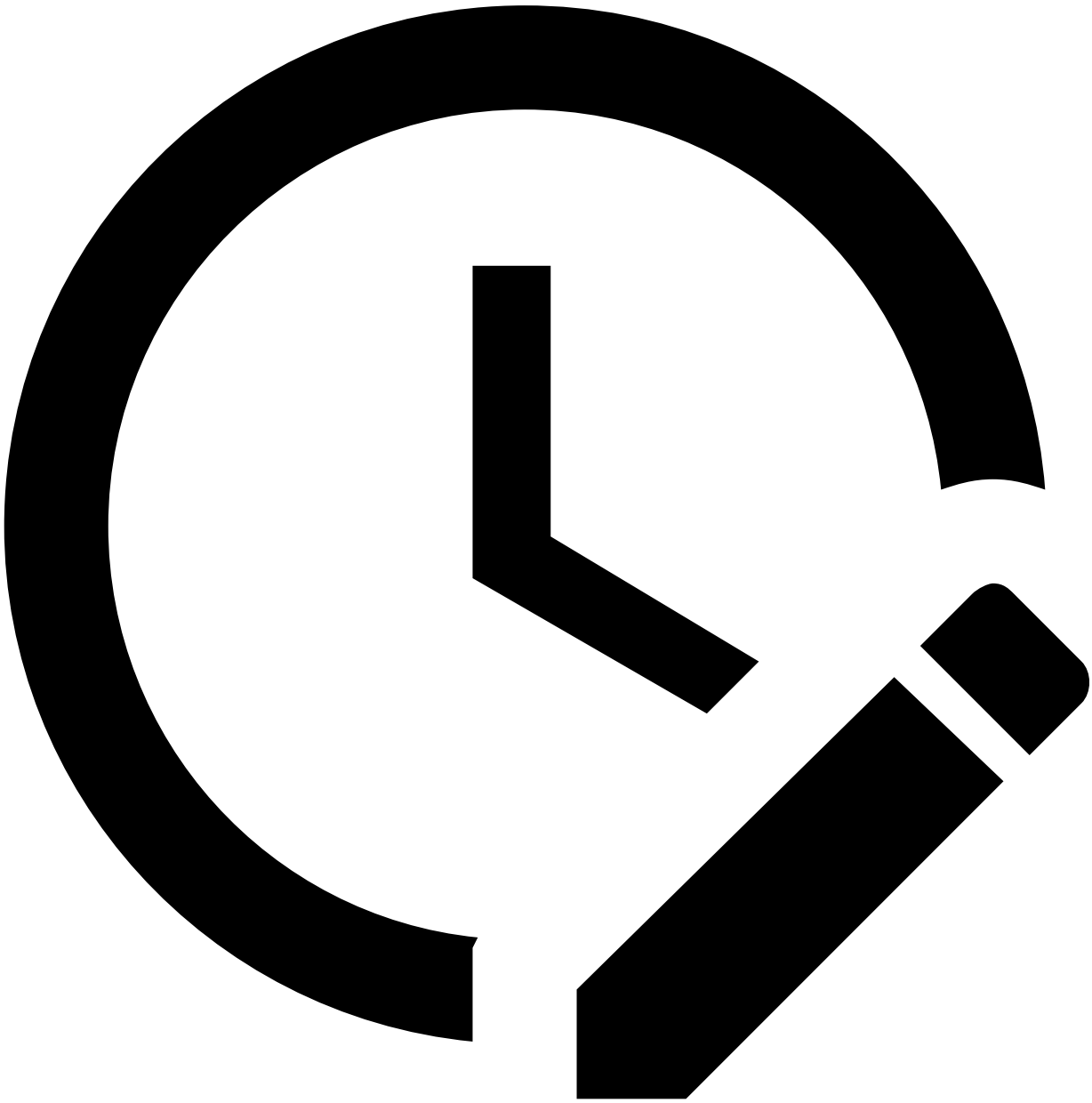
2.3.2 Przykład (Node + socket

i

o

)

```
io.use(authMiddleware);
io.on("connection", (s) => {
  s.join(`user:${s.user.id}`);
  s.on("cmd", (payload) => {
    /* validate + run */
  });
});
```



3 października 2025

2.4 OTUI — pods

t

a

w

y

> Info

OTUI to deklaratywne layouty interfejsu użytkownika.

2.4.1 Przykład layo

u

t

u

```
Panel
  id: main
  anchor: top left
  size: 400 300

Label
  text: "Status: OK"
  anchors.centerIn: parent
```

2.4.2 Zdarzenia / wiąza

n

i

a

- Właściwości elementów można powiązać z danymi (np. przez Lua).
- Aktualizacje push przez eventy modułów.

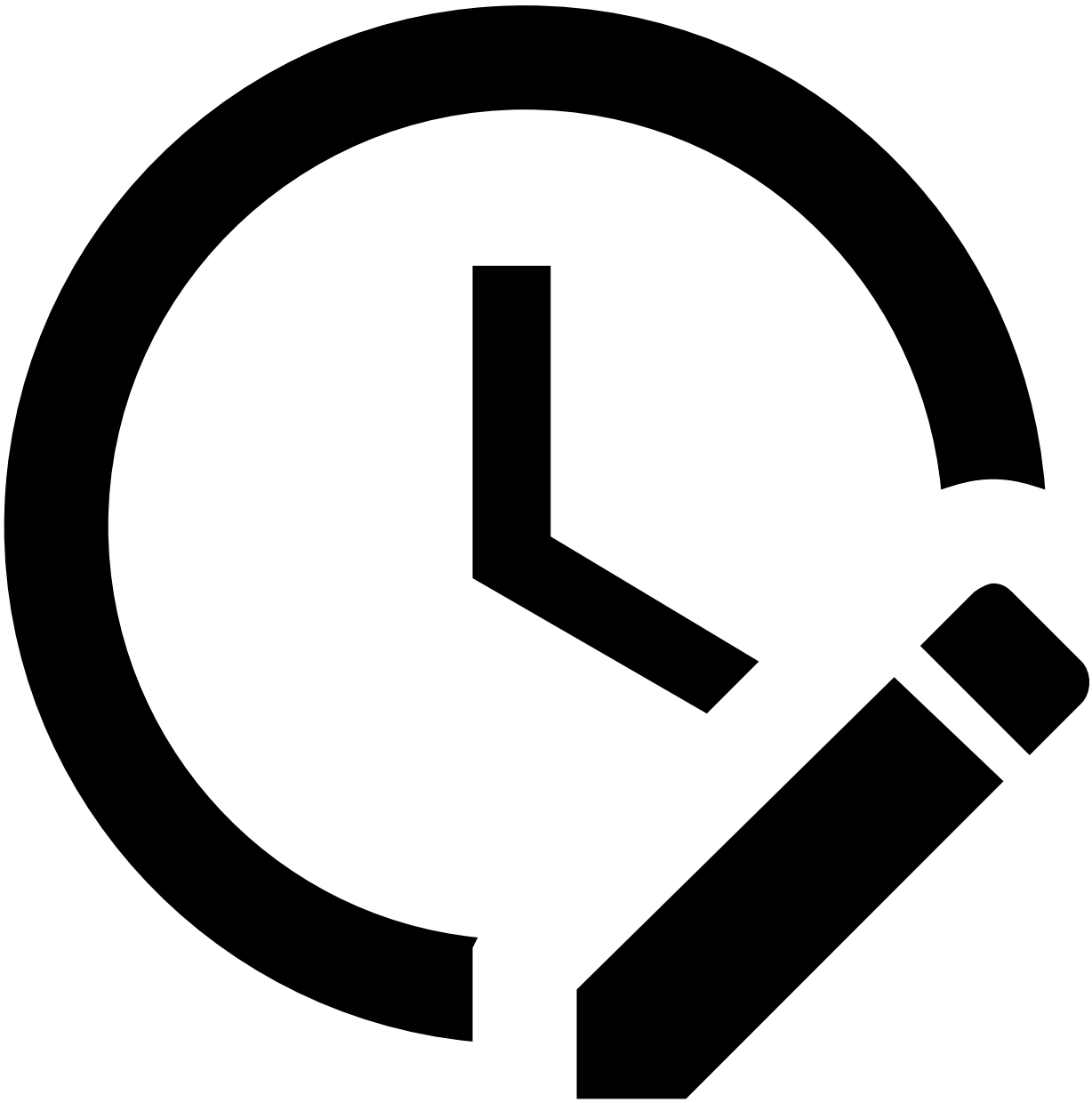
2.4.3 Wskazó

w

k

i

- Trzymaj layouty w `layouts/*`.
- Styluj wspólnymi klasami, nie inline.

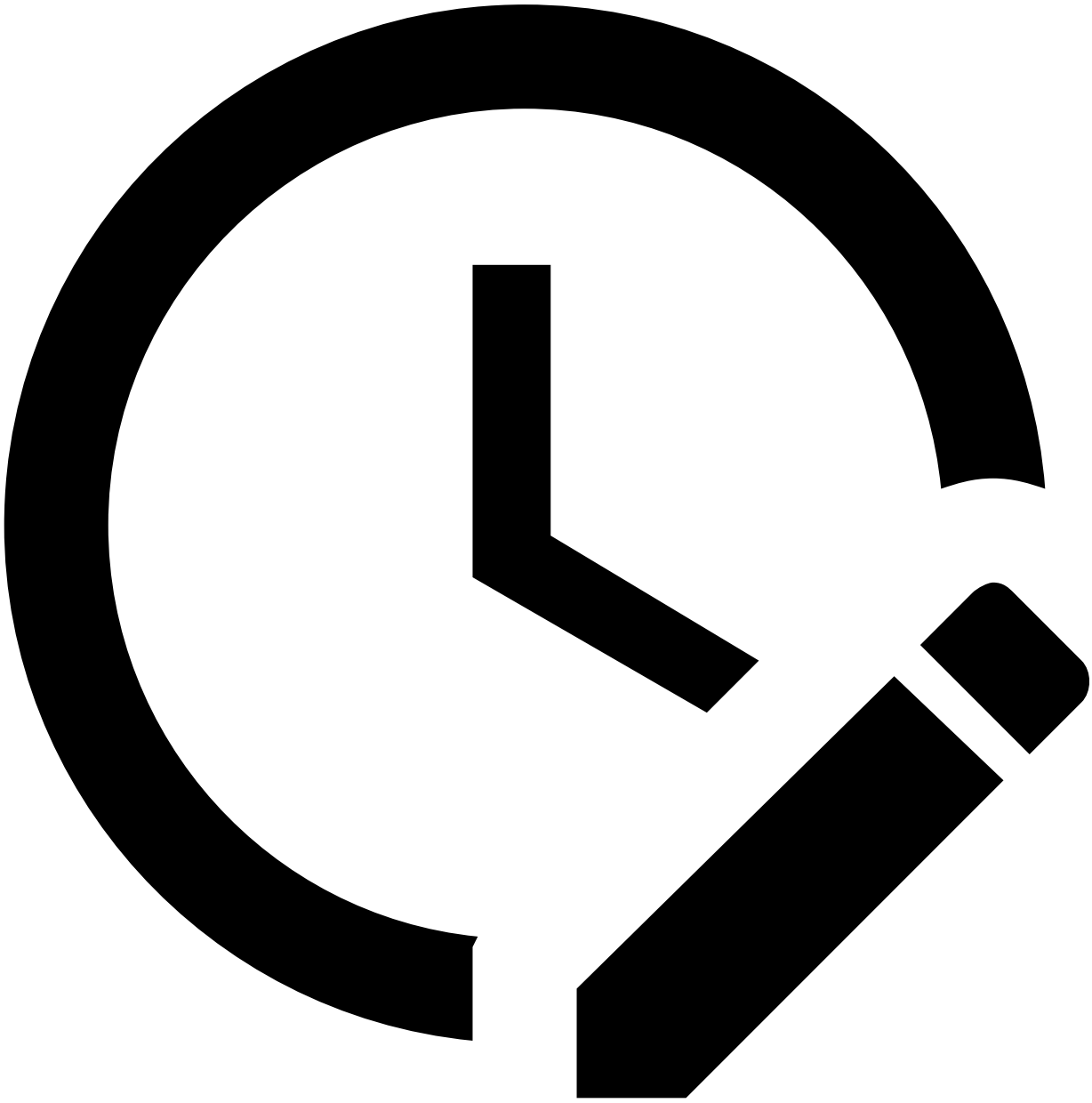


3 października 2025

3. Build

3.1 Build — Windows

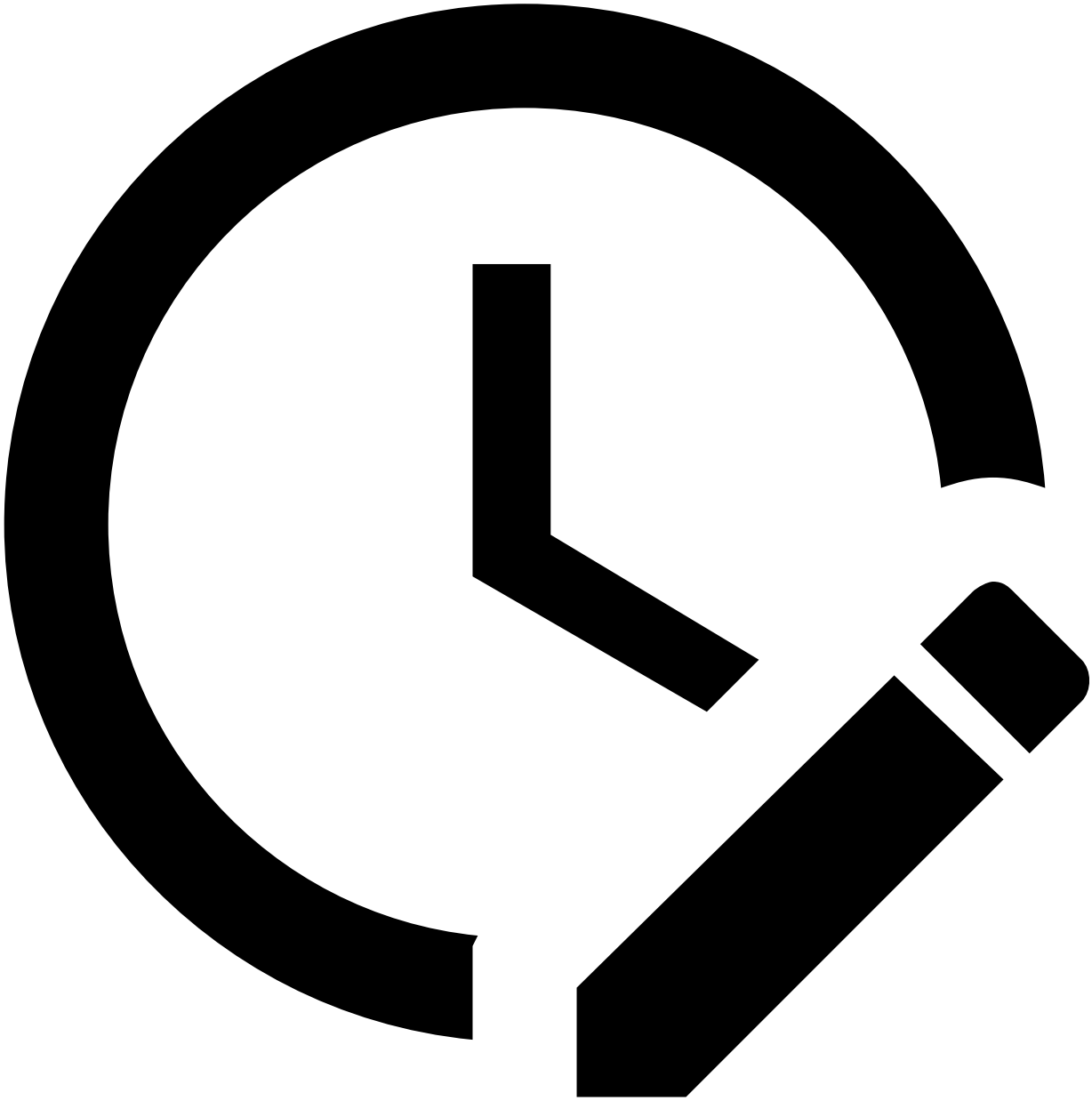
1. Zainstaluj **Visual Studio 2019 + vcpkg**.
2. W vcpkg doinstaluj zależności (patrz README projektu).
3. Kompilacja wg instrukcji repo (tu wklej swoje komendy 1:1).



3 października 2025

3.2 Build — Linux

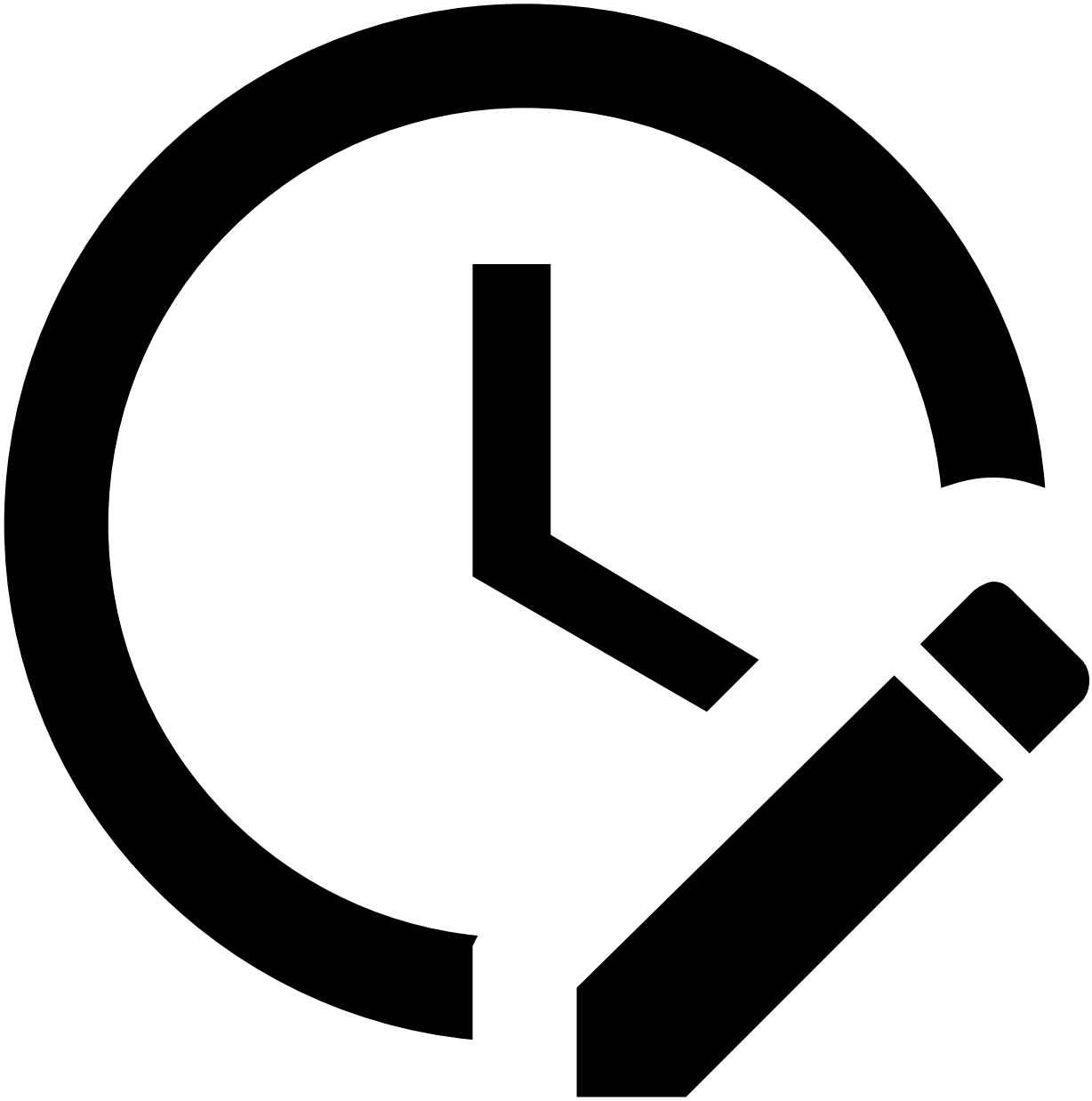
1. Zależności systemowe (gcc/clang, cmake, itp.)
2. Kroki kompilacji.



3 października 2025

3.3 Build — Android

1. NDK/SDK, Java, skrypty assets.
2. Kroki kompilacji/apk.



3 października 2025

4. Lua

4.1 Lua API (przykł

a

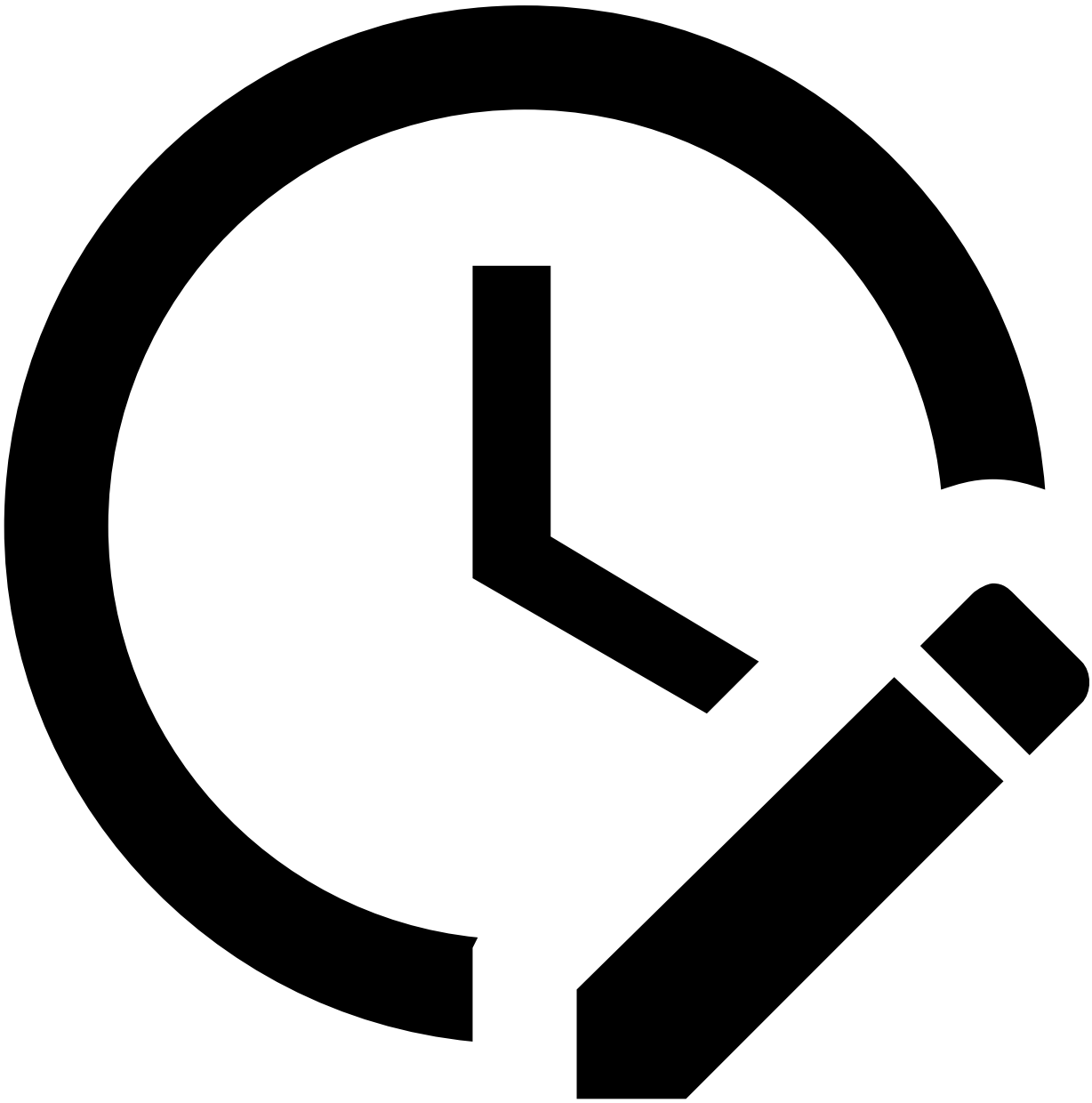
d

y

)

```
-- przykład zdarzenia
onTalk(function(name, level, mode, text)
  -- ...
end)
```

Docelowo: generowane referencje (LDoc/EmmyLua) + przykłady.



3 października 2025

4.2 Lua — style g

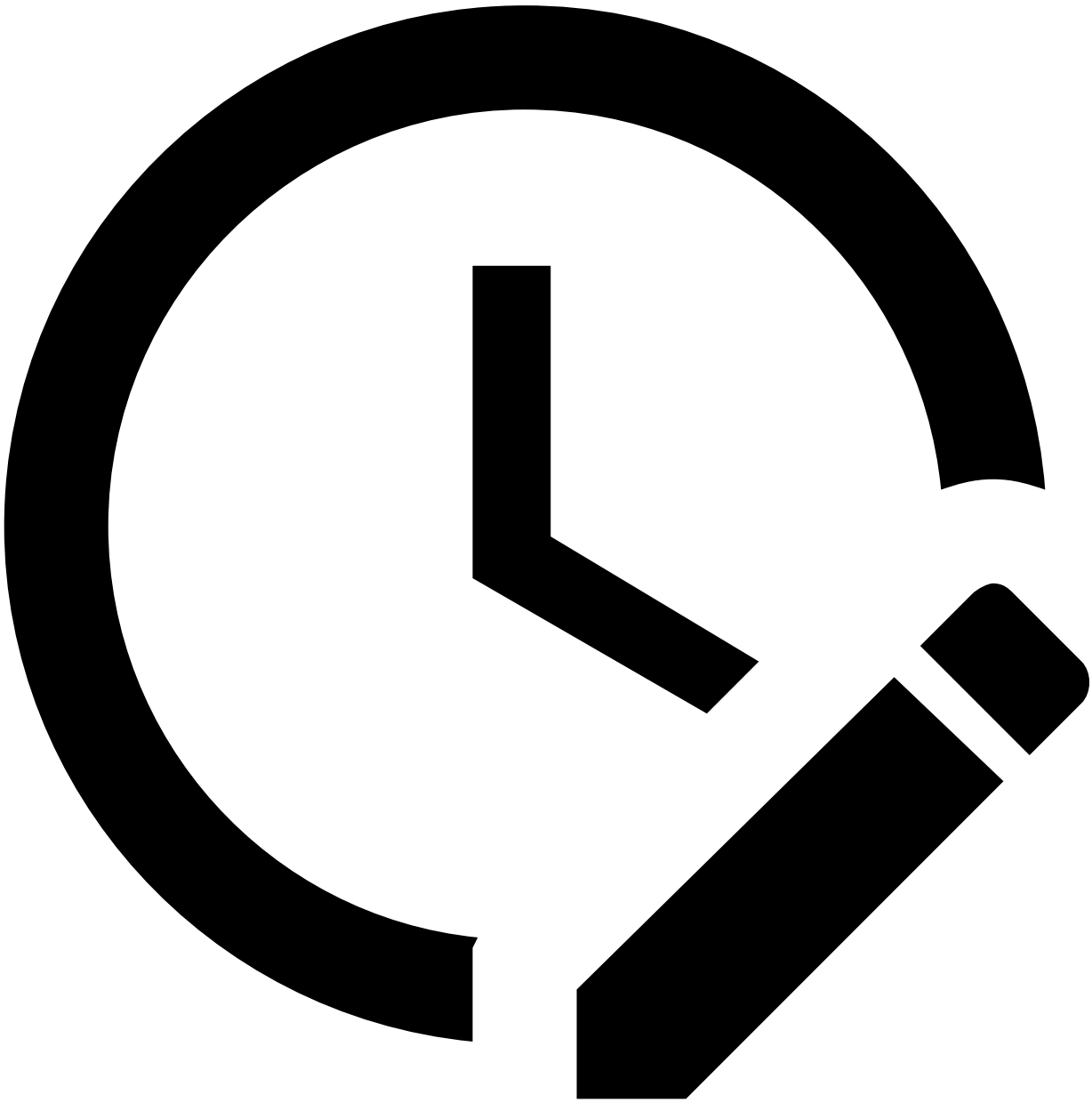
u

i

d

e

- snake_case dla zmiennych i funkcji.
- Moduły zwracają tabelę publicznego API.
- Brak efektów ubocznych w `require`.
- Pliki < 300 linii, funkcje < 50 linii.
- Obsługa błędów: `pcall/xpcall` dla krytycznych ścieżek.
- Logowanie warunkowe przez `DEBUG` flagę.



3 października 2025

5. C++

5.1 C++ — prze

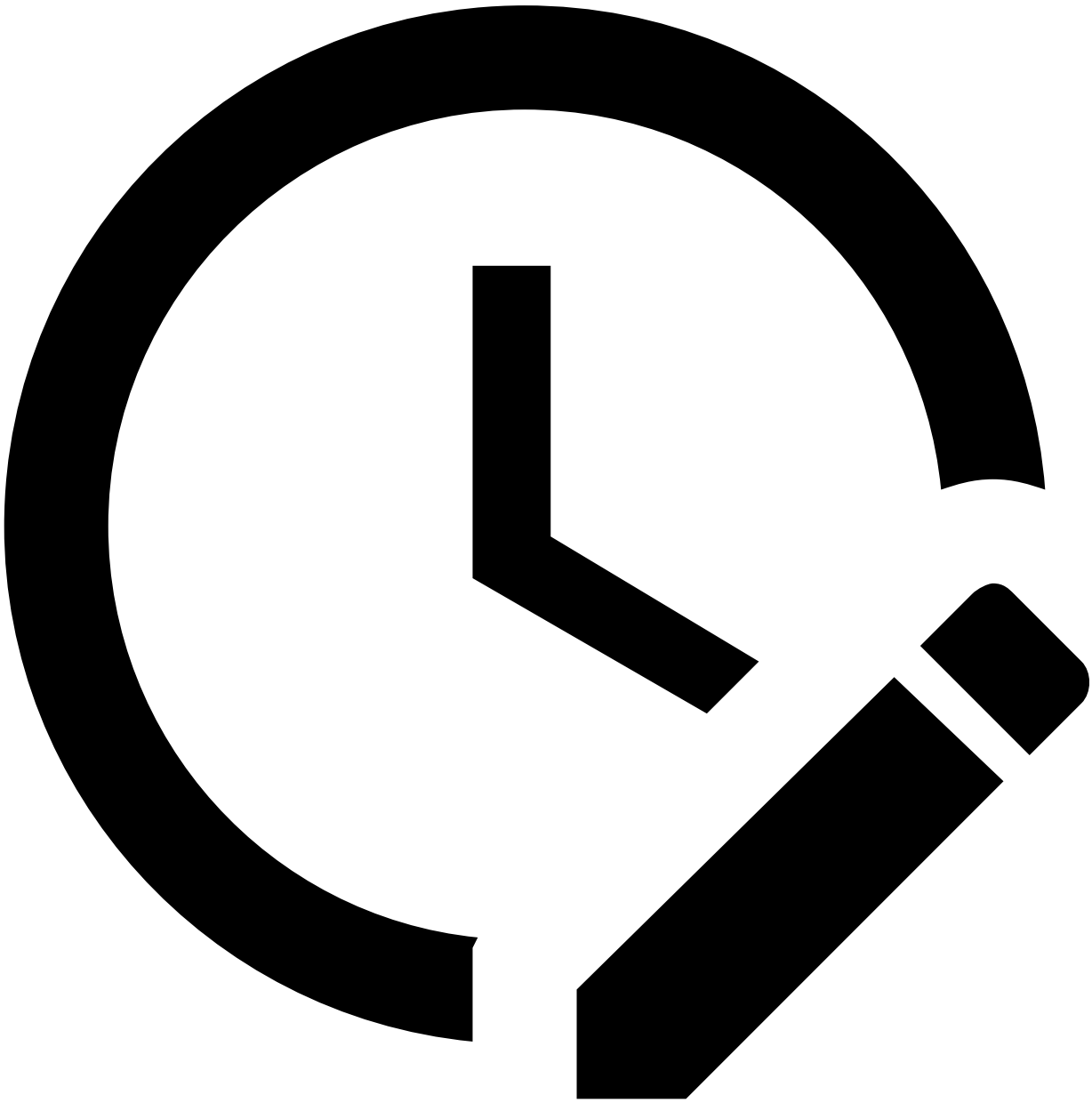
g

l

ą

d

- Struktura katalogów
- Kluczowe klasy/entrypoints
- Jak rozszerzać (hooki, interfejsy)



3 października 2025

6. Narzę

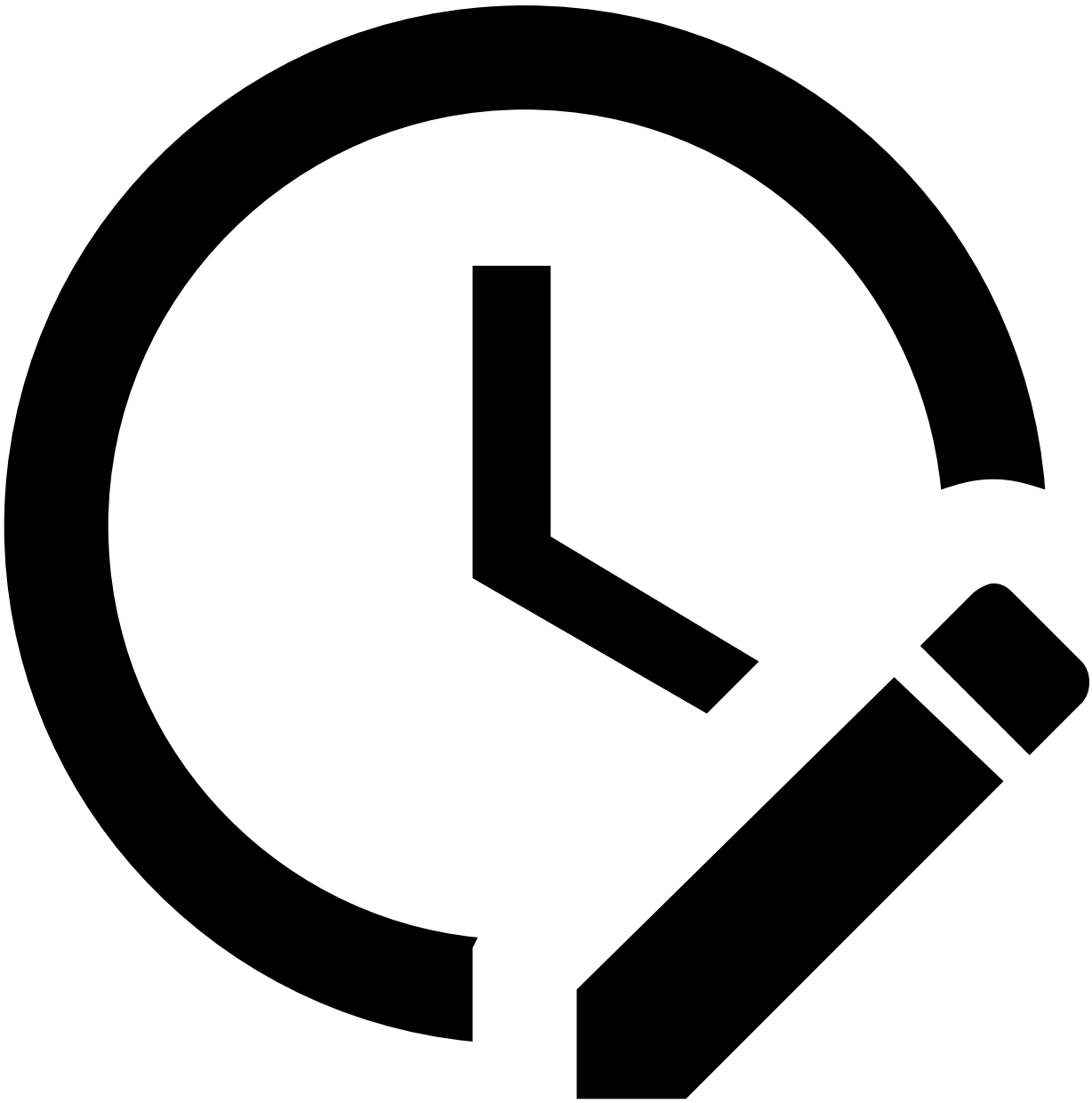
d

z

i

a

- Skrypty pomocnicze znajdziesz w `tools/` repo.
- Uzupełnij listę narzędzi po weryfikacji w repo.



3 października 2025

7. Deweloperzy

7.1 Contribu

t

i

n

g

1. Fork → branch `feature/…`.
2. Commity: konwencja `type(scope): msg` (np. `docs(build): …`).
3. PR z krótkim opisem i screenami.
4. Review: 1 LGTM + zielone CI.

7.1.1 St

y

l

e

- Lua: patrz *Lua → Style guide*.
- C++: clang-format (domyślny styl projektu).

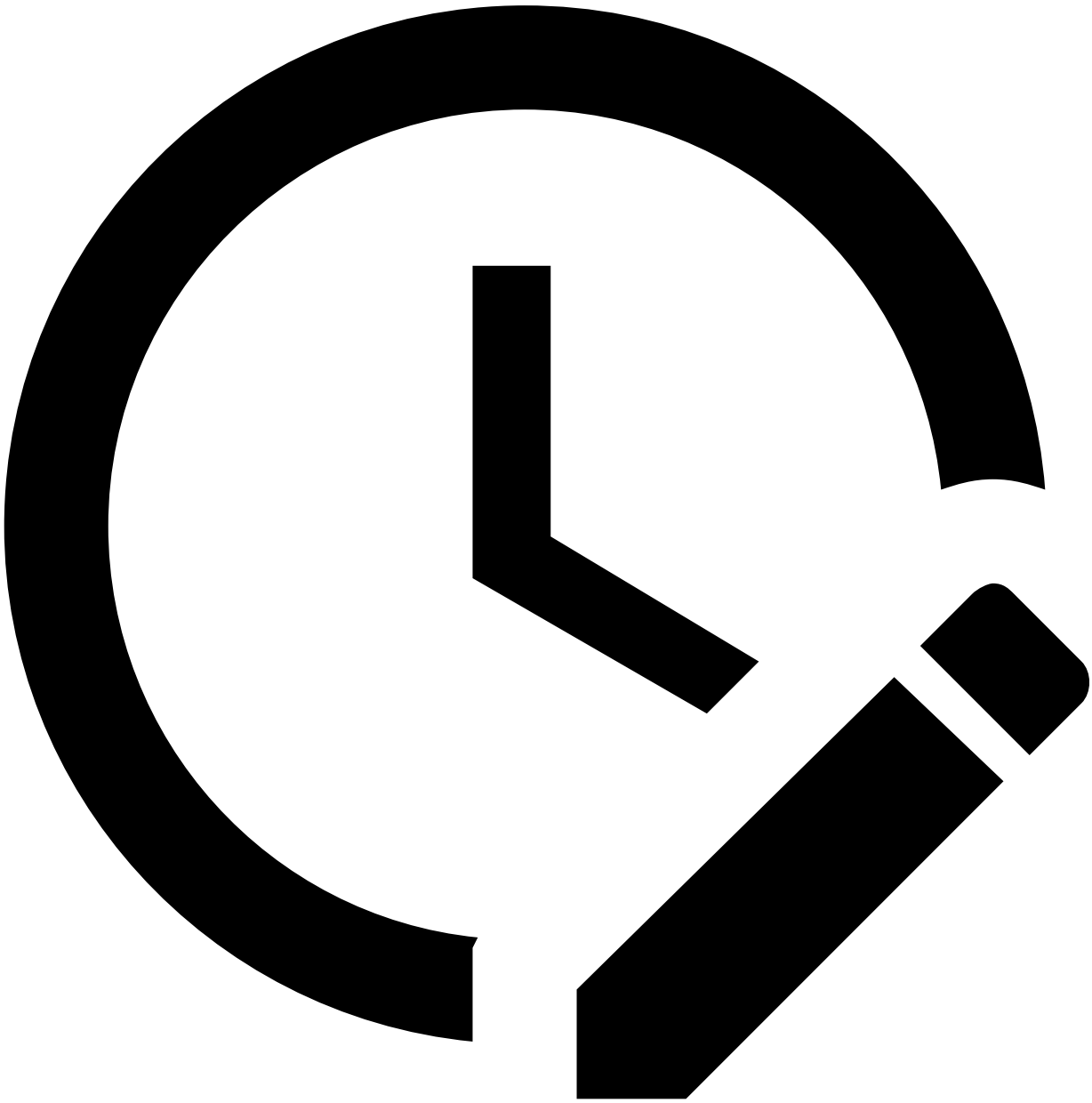
7.1.2 Commity dokumenta

c

j

i

- Zmiany w `docs/**` nie uruchamiają CMake (paths-ignore).



3 października 2025

7.2 T

e

s

t

y

- Lua: testy jednostkowe (busted) — folder `spec/`.
- C++: GoogleTest (jeśli dostępne).
- E2E: scenariusze ręczne + checklisty w PR.

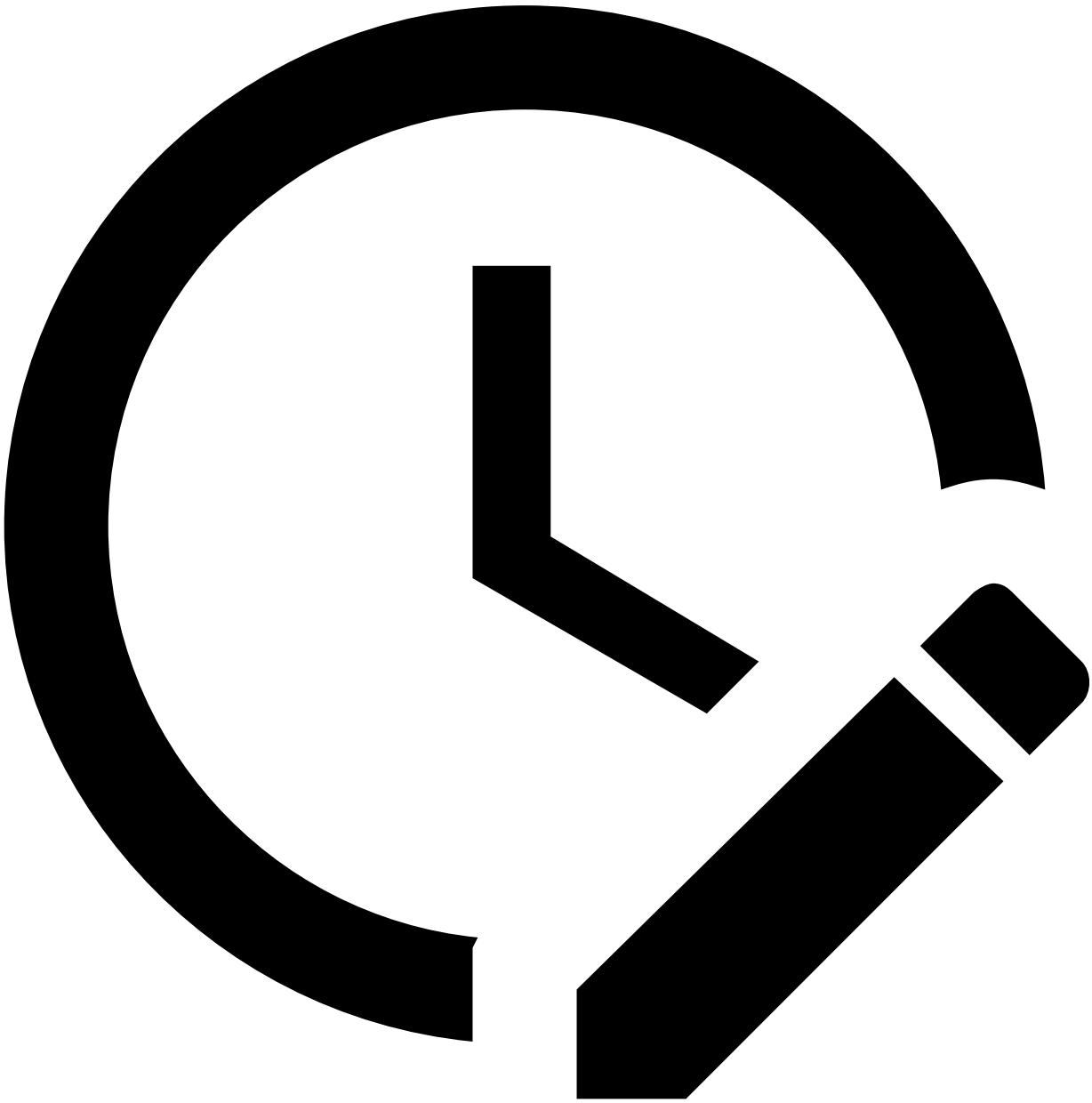
7.2.1 Raport bł

ę

d

u

- Kroki odtworzenia, logi, wersja, OS.



3 października 2025

7.3 Rel

e

a

s

e

1. Bump wersji (semver) / tag.
2. Changelog z PR od ostatniego taga.
3. Build artefaktów (CI).
4. Publikacja + checksumy.

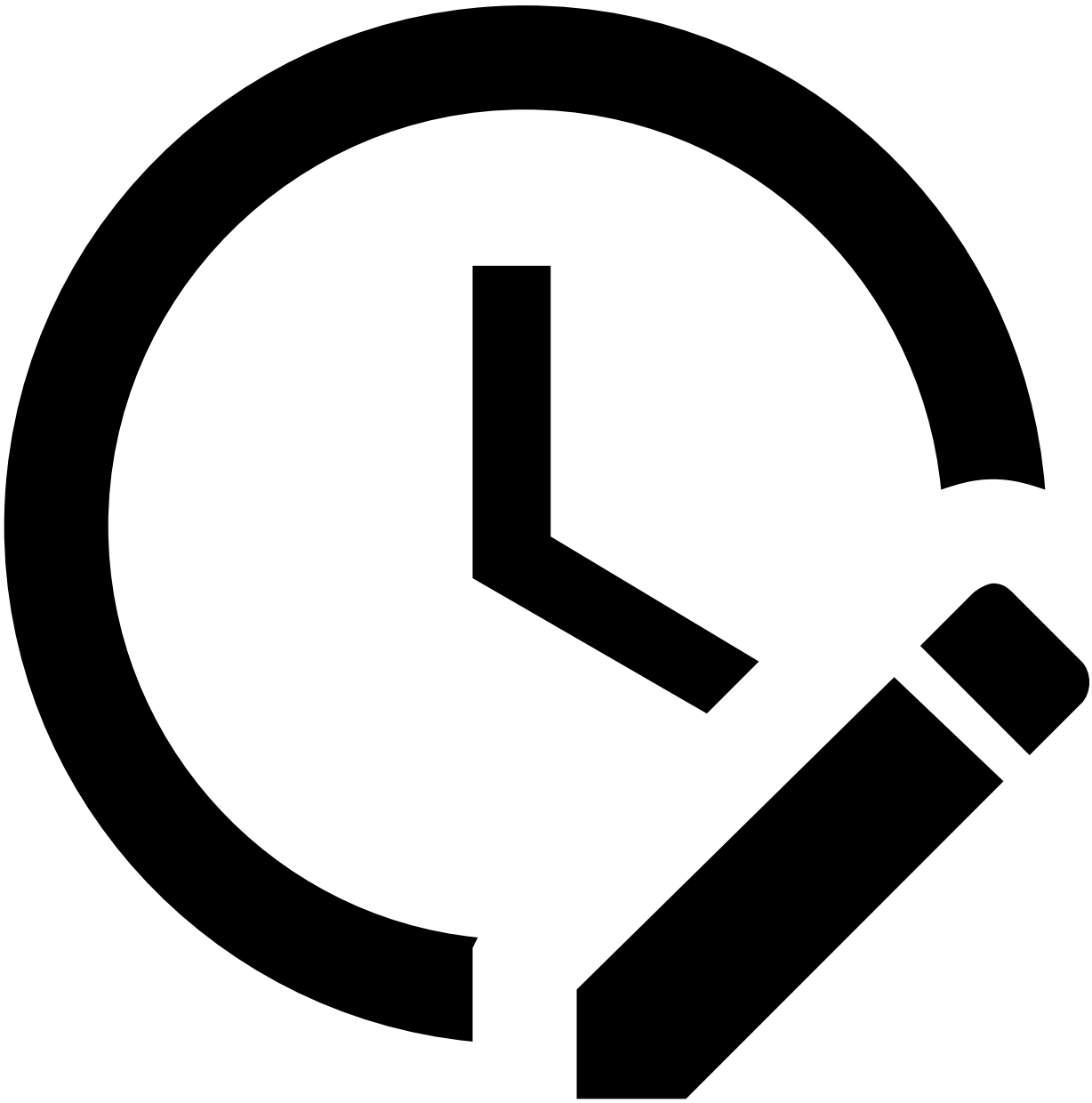
7.3.1 Wersjonowanie dokumenta

c

j

i

- Docsy budują się z gałęzi `master`.



3 października 2025

7.4 Secu

r

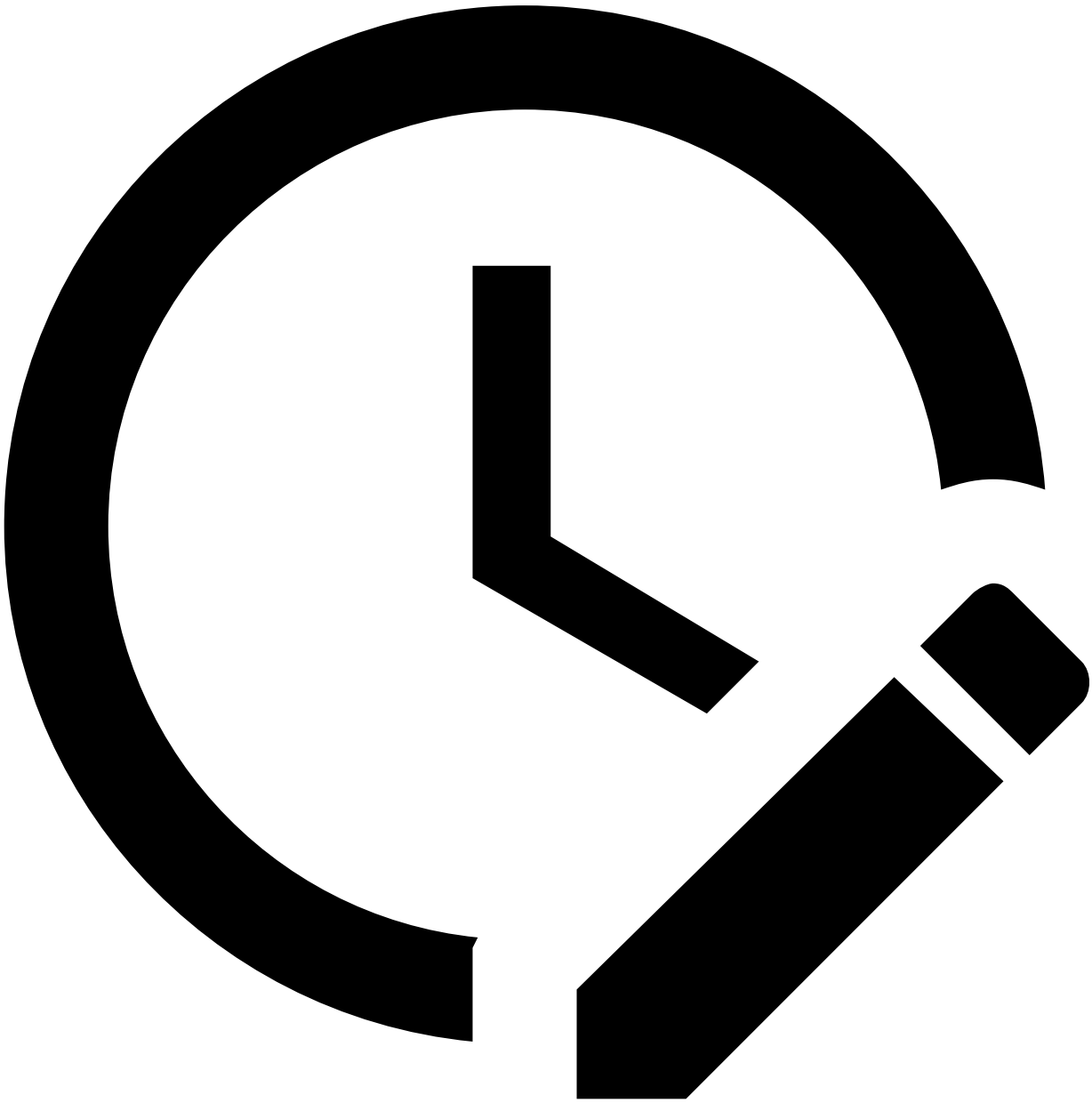
i

t

y

- HTTPS/WSS, HSTS, twardy CSP i CORS.
- Auth: krótkie JWT + refresh, RBAC.
- WS: origin check, limit bufora, ping.
- Walidacja payloadów (Zod/JSON Schema).
- Sekrety nigdy w repo (env w CI).

Zgłoszenia luk: security@twojadomena.example



3 października 2025

7.5 Troubleshoo

t

i

n

g

7.5.1 Bu

i

l

d

- Brak zależności vcpkg → zainstaluj pakiety i przebuduj cache.

7.5.2 Andr

o

i

d

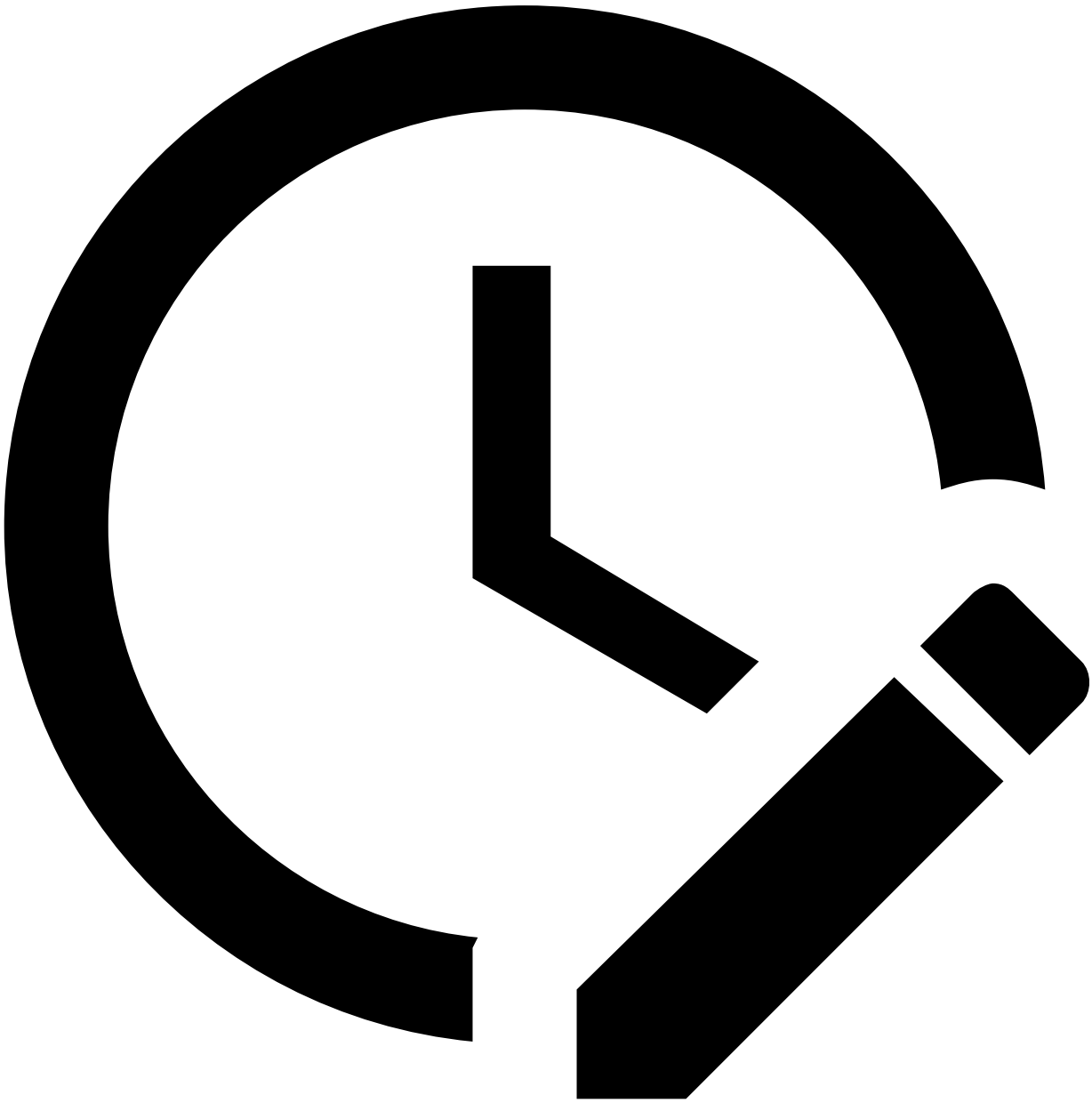
- Błąd NDK: sprawdź wersję i ścieżki SDK/NDK.

7.5.3

W

S

- Rozłączenia: sprawdź pingTimeout/pingInterval oraz proxy.



3 października 2025

7.6

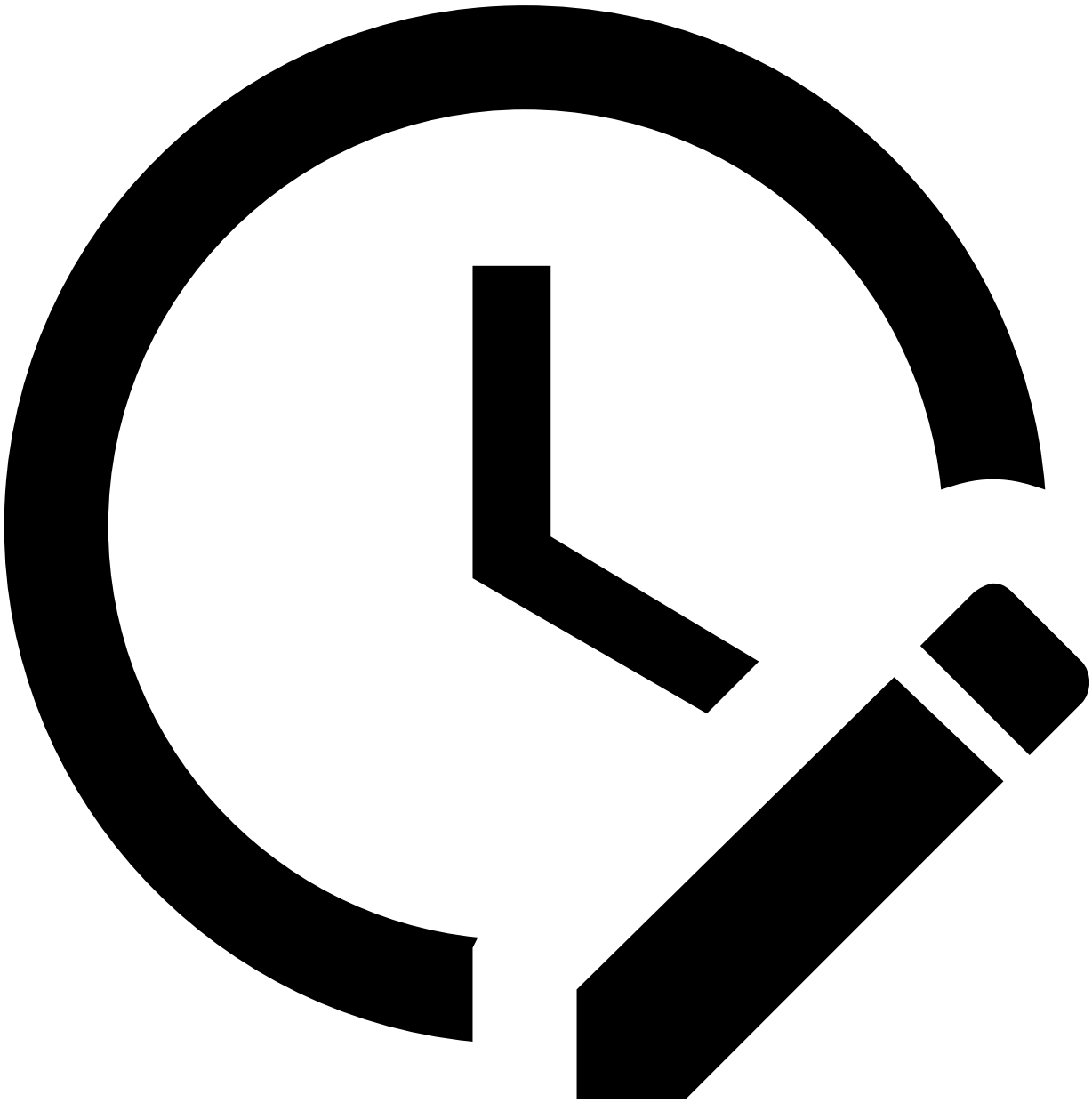
F

A

Q

Czy mogę używać swoich modułów? Tak, wrzuć do `modules/<nazwa>` i zarejestruj w konfiguracji.

Czy działa na Linux/Windows? Tak — patrz sekcja *Build*.



3 października 2025

7.7 Sło

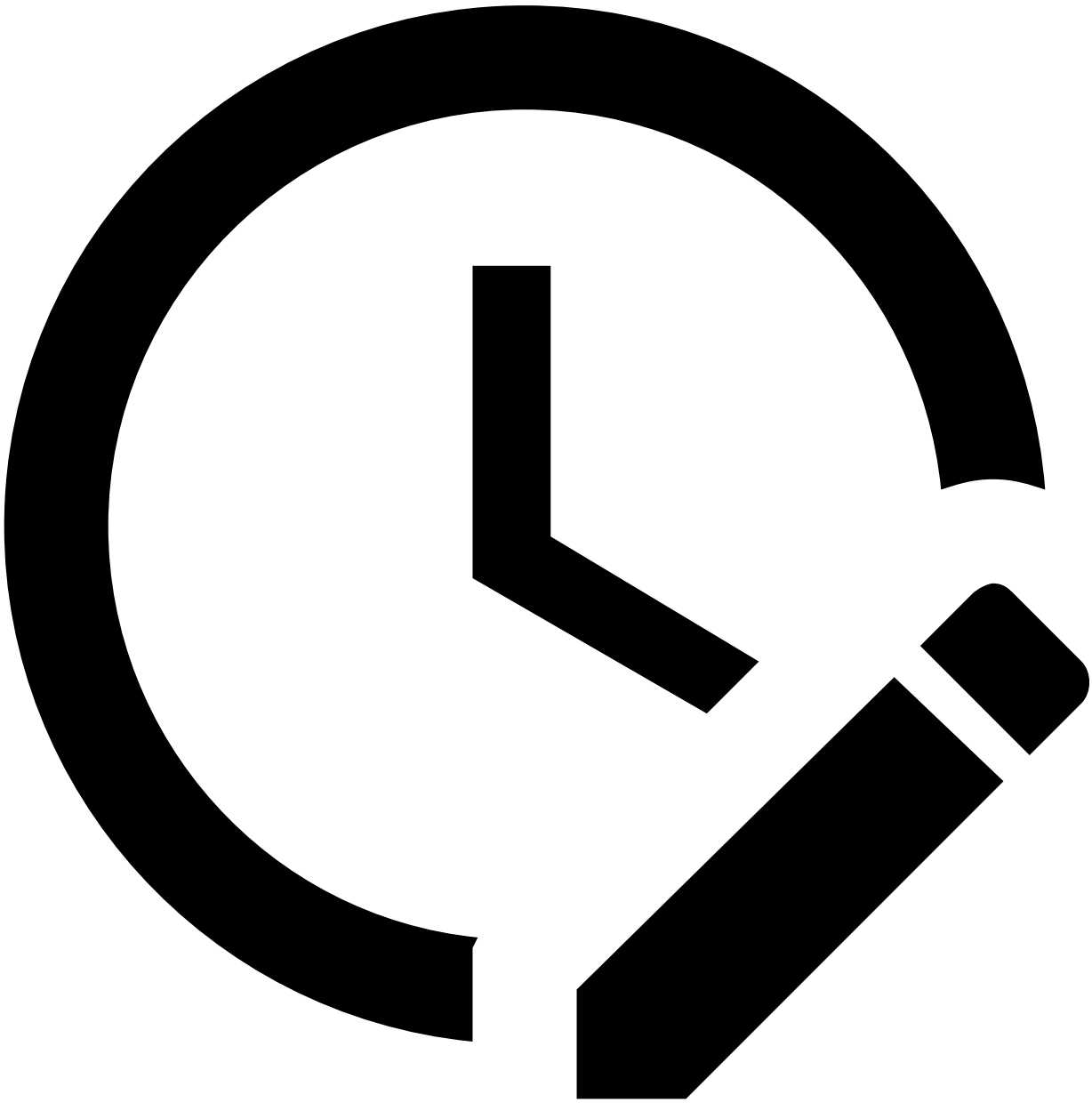
w

n

i

k

- **OTUI** — język layoutów UI.
- **vBot** — moduły Lua automatyzujące zachowania.
- **RAG** — Retrieval Augmented Generation (wyszukiwanie + LLM).



3 października 2025

7.8 Roa

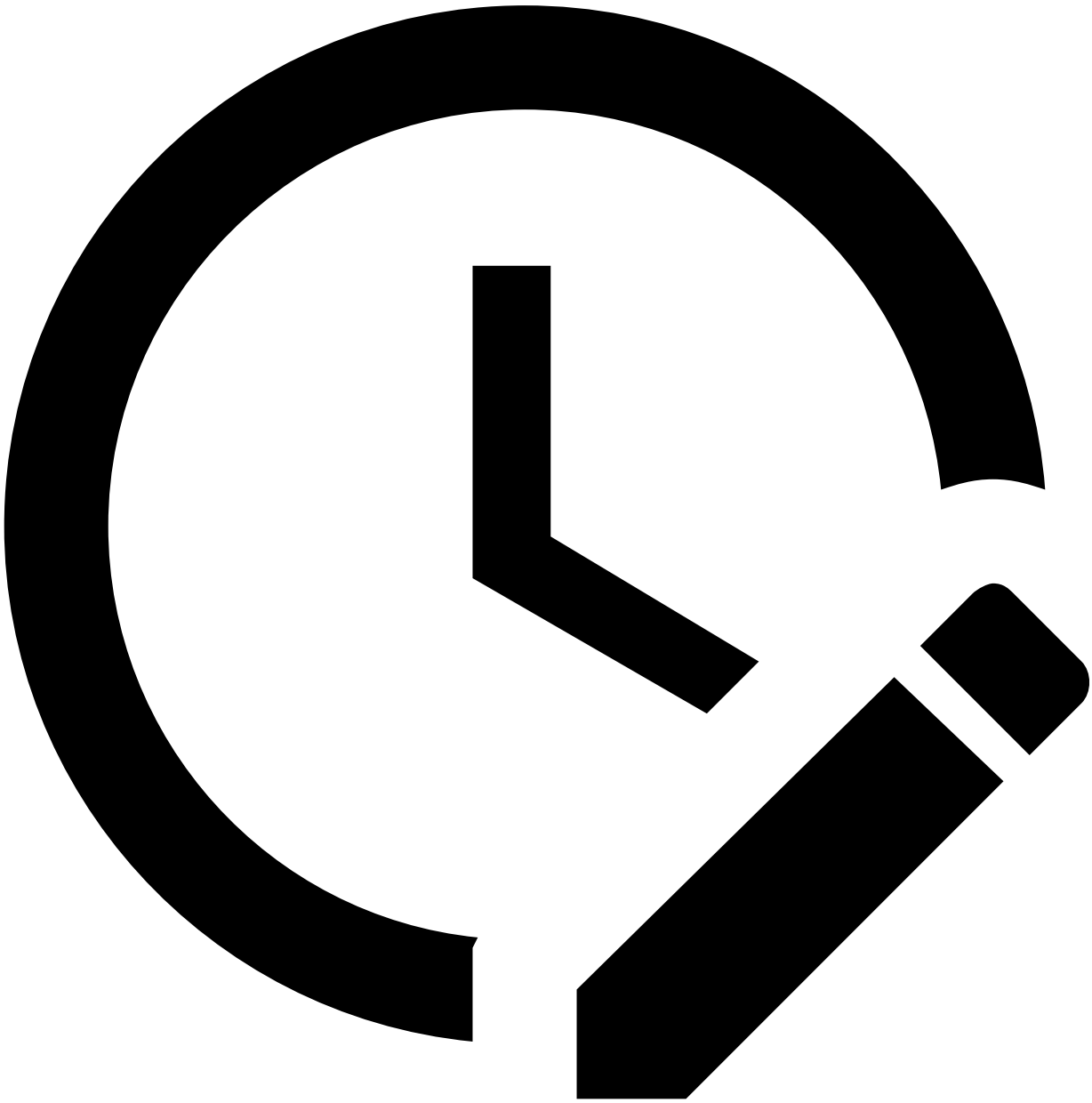
d

m

a

p

- [ ] Pełna referencja API Lua (LDoc).
- [ ] Generowanie Doxygen dla C++ i wpięcie do docs.
- [ ] Więcej przykładów OTUI.
- [ ] Testy E2E dashboardu.



3 października 2025