# OTCv8 Dev Docs

# Start

# 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).

!!! info “Baza wiedzy dla edytora”

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

# Architektura

# Architekt

u

r

a

(

s

k

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

# Moduły (vBot)

# Moduły (vBot) —

p

r

z

e

w

o

d

n

i

k

!!! info “Cel”

Jak tworzyć i ładować moduły Lua dla klienta OTCv8.

## Strukt

u

r

a

m

o

d

u

ł

u

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

## Minimalny m

o

d

u

ł

(

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

## Rejestrowanie zdarzeń (

p

r

z

y

k

ł

a

d

)

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

## Kon

f

i

g

u

r

a

c

j

a

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

## D

e

b

u

g

/

l

o

g

i

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

## Dobr

e

p

r

a

k

t

y

k

i

* Nazwy przestrzeni modułu (my\_module.\*).
* Brak efektów ubocznych przy require.
* Komendy eksportuj jawnie (np. M.start, M.stop).

# Realtime (WebSocket)

# Realtime (

W

e

b

S

o

c

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

Z

a

s

a

d

y

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

## Przykład (Node +

s

o

c

k

e

t

i

o

)

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

# OTUI

# OTU

I

—

p

o

d

s

t

a

w

y

!!! info

OTUI to deklaratywne layouty interfejsu użytkownika.

## Przykł

a

d

l

a

y

o

u

t

u

Panel  
 id: main  
 anchor: top left  
 size: 400 300  
  
Label  
 text: "Status: OK"  
 anchors.centerIn: parent

## Zdarzenia

/

w

i

ą

z

a

n

i

a

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

W

s

k

a

z

ó

w

k

i

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

# Windows

# 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).

# Linux

# Build — Linux

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

# Android

# Build — Android

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

# API (przykłady)

# Lua API (

p

r

z

y

k

ł

a

d

y

)

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

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

# Style guide

# Lua —

s

t

y

l

e

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.

# Przegląd

# C+

—

p

r

z

e

g

l

ą

d

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

# Narzędzia

N

a

r

z

ę

d

z

i

a

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

# Contributing

# Co

n

t

r

i

b

u

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.

S

t

y

l

e

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

## Commity dok

u

m

e

n

t

a

c

j

i

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

# Testy

T

e

s

t

y

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

## Ra

p

o

r

t

b

ł

ę

d

u

* Kroki odtworzenia, logi, wersja, OS.

# Release

R

e

l

e

a

s

e

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

## Wersjonowanie dok

u

m

e

n

t

a

c

j

i

* Docsy budują się z gałęzi master.

# Security

S

e

c

u

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](mailto:security@twojadomena.example)

# Troubleshooting

# Troub

l

e

s

h

o

o

t

i

n

g

B

u

i

l

d

* Brak zależności vcpkg → zainstaluj pakiety i przebuduj cache.

A

n

d

r

o

i

d

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

W

S

* Rozłączenia: sprawdź pingTimeout/pingInterval oraz proxy.

# FAQ

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*.

# Słownik

S

ł

o

w

n

i

k

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

# Roadmap

R

o

a

d

m

a

p

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

# Pełne API (auto)

# OTCv8 – Pełn

e

A

P

I

Wersja specyfikacji: 1.0 Status: **draft** (uzupełnij pola TODO: realnymi nazwami/argumentami z kodu).

## 0. Konwe

n

c

j

e

* **Typy**: string, number, boolean, table, nil.
* **Czas**: UNIX ms (number).
* **Błędy**: kody E\_\* + opis; w WS pole error.
* **Nazewnictwo**:
  + Lua: snake\_case
  + Eventy: dot.case (metrics.update, cmd.result)
  + OTUI: id unikalne, class do stylów.

## 1. Runtime Lua / Moduły

v

B

o

t

### 1.1. Cykl życia mo

d

u

ł

u

-- modules/<name>/init.lua  
local M = {}  
  
-- Wywoływane przy starcie modułu  
function M.start(ctx) end  
  
-- Wywoływane przy zatrzymaniu modułu  
function M.stop(ctx) end  
  
-- (opcjonalnie) okresowe ticki  
function M.tick(ctx, dt\_ms) end  
  
return M

**ctx (context)** – most do UI/WS/IO:

| Metoda | Opis | Przykład |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ctx.emit(target, payload) | Publikuj zdarzenie do UI/mostu. | ctx.emit("status", {text="OK"}) |  |  |  |
| ctx.on(event, fn) | Subskrypcja lokalnego eventu. | ctx.on("config.changed", fn) |  |  |  |
| ctx.log(msg, level?) | Log ("debug" | "info" | "warn" | "error"). | ctx.log("start","info") |  |  |  |
| ctx.set\_timeout(ms, fn) | Jednorazowy timer; zwraca id. |  |  |  |  |
| ctx.clear\_timeout(id) | Anuluj timer. |  |  |  |  |
| ctx.storage.get(key) | Odczyt z pamięci modułu. |  |  |  |  |
| ctx.storage.set(key,val) | Zapis do pamięci modułu. |  |  |  |  |
| ctx.ws.send(type, data) | Wyślij wiadomość WS (patrz §3). | ctx.ws.send("telemetry.push",{hp=100}) |  |  |  |

TODO: dopisz rzeczywiste metody ctx z projektu (HTTP? FS? IPC?).

### 1.2. Zdarzenia globalne Lua (emiter siln

i

k

a

)

Rejestrujesz globalnie (przed startem lub w M.start). **Silnik emituje** m.in.:

-- Chat/komunikacja  
onTalk(function(name, level, mode, text) end) -- TODO: potwierdź argumenty  
  
-- Obiekty  
onCreatureAppear(function(creature) end)  
onCreatureDisappear(function(creature) end)  
onCreatureHealthChange(function(creature, hp, hpmax) end)  
  
-- Wejście/użytkownik  
onKeyDown(function(key) end)  
onKeyUp(function(key) end)  
onMouseEvent(function(x, y, btn, state) end)  
  
-- Czas  
every(ms, function() end) -- helper okresowy  
schedule(ms, function() end) -- jednorazowy  
cancel(timer\_id)

TODO: uzupełnij pełną listę eventów i ich parametry.

### 1.3. Błędy (

L

u

a

)

* E\_ARG – zły typ/zakres argumentu
* E\_STATE – nieprawidłowy stan (np. brak UI id)
* E\_TIMEOUT – operacja przekroczyła czas
* E\_INTERNAL – błąd wewnętrzny

## 2. OTUI (UI Bri

d

g

e

)

### 2.1. Identyfikacja i wiąz

a

n

i

a

* Każdy element ma id (unikalny w layoucie).
* Zmiana właściwości przez emit:

-- Tekst w Label#status  
ctx.emit("status", { text = "Running" })

**Mapowanie domyślne (bridge po id):**

| Emisja target | Element OTUI (przykład) | Pola |
| --- | --- | --- |
| "status" | Label#status | text, color?, visible? |
| "hp\_bar" | ProgressBar#hp\_bar | value, max?, color? |
| "list" | ListView#list | items: string[] |

TODO: dopisz Twoje id → obsługiwane pola.

### 2.2. Layout – przy

k

ł

a

d

Panel  
 id: main  
 size: 400 240  
 anchor: top left  
  
Label  
 id: status  
 text: "init"  
 anchors.centerIn: parent

### 2.3. Zdarzenia z UI d

o

L

u

a

UI może wywołać event do Lua:

ctx.on("ui.click.status", function(args) end)

TODO: lista eventów z UI (klik, input, select) i payloady.

## 3. WebSocket (proto

k

ó

ł

)

### 3.1. Kanał i

t

r

y

b

* Transport: wss://<host>/ws (JSON UTF-8).
* Autoryzacja: JWT w Authorization: Bearer <token> lub cookie sesji.
* Origin allowlist; pingTimeout; rate-limit i maks. payload (np. 32KB).

### 3.2. Typy wiadom

o

ś

c

i

#### Serwer → Klient (dashbo

a

r

d

)

type ServerEvent =  
 | { type: "metrics.update"; ts: number; payload: { hp: number; mp: number; [k: string]: number } }  
 | { type: "log.line"; ts: number; level: "debug" | "info" | "warn" | "error"; msg: string }  
 | { type: "char.info"; ts: number; payload: { name: string; level: number; voc: string } }  
 | { type: "cmd.result"; ts: number; id: string; ok: boolean; error?: string; data?: any };

#### Klient → Se

r

w

e

r

type ClientEvent =  
 | { type: "cmd"; id: string; name: "START" | "STOP" | "RELOAD"; args?: any }  
 | { type: "subscribe"; topics: string[] } // np. ["metrics.\*","log.\*"]  
 | { type: "settings.update"; patch: Record<string, any> };

**Przykład**:

{ "type": "cmd", "id": "abc123", "name": "START", "args": { "profile": "pvp" } }

### 3.3. JSON Schema (walidacja na serwe

r

z

e

)

schemas/ws/cmd.schema.json

{  
 "$id": "ws/cmd.schema.json",  
 "type": "object",  
 "required": ["type", "id", "name"],  
 "properties": {  
 "type": { "const": "cmd" },  
 "id": { "type": "string", "minLength": 1 },  
 "name": { "type": "string", "enum": ["START", "STOP", "RELOAD"] },  
 "args": { "type": "object", "additionalProperties": true }  
 },  
 "additionalProperties": false  
}

TODO: dodaj schematy metrics.update, log.line, settings.update, …

### 3.4. Błę

d

y

W

S

* 401 – brak/niepoprawny token
* 403 – brak uprawnień (RBAC)
* 429 – limit
* 4401 (app) – walidacja payloadu (error: "E\_SCHEMA")

## 4. C++ / rozszerz

e

n

i

a

**Szkic struktur (dopasuj do projektu):**

// include/otcv8/api.h  
struct OTEvent {  
 const char\* type; // "metrics.update"  
 uint64\_t ts\_ms;  
 const char\* json; // payload JSON  
};  
  
using OttoEmitFn = void(\*)(const OTEvent\* ev); // emit from C++ -> Lua/UI

**Punkty integracji:**

* Rejestracja hooków (zdarzenia gry → Lua)
* Export funkcji do Lua (np. say, use\_item, …)
* Konwencja ładowania modułów C++: dll/so w modules/<name>/bin/.

TODO: wstaw prawdziwe nagłówki i entrypointy.

## 5. Błędy i

k

o

d

y

| Kod | Warstwa | Opis |
| --- | --- | --- |
| E\_ARG | Lua | nieprawidłowy argument |
| E\_STATE | Lua | zły stan (np. brak UI id) |
| E\_WS | WS | połączenie zerwane |
| E\_SCHEMA | WS | payload niezgodny ze schematem |
| E\_IO | C++/FS | błąd I/O |
| E\_INT | All | błąd wewnętrzny |

Format odpowiedzi błędu (WS):

{  
 "type": "cmd.result",  
 "ts": 1712345678,  
 "id": "abc",  
 "ok": false,  
 "error": "E\_SCHEMA",  
 "data": { "path": "/args/profile" }  
}

## 6. Bezpieczeństwo (sk

r

ó

t

)

* HTTPS/WSS, HSTS, twardy **CSP**.
* Token JWT krótki + refresh, **RBAC**.
* Check **Origin**, **rate-limit**, **message size**.
* Sekrety poza repo (env/CI), brak kluczy w repo.
* Logowanie dostępu i audyt komend WS.

## 7. Versio

n

i

n

g

* SemVer API: major.minor.patch.
* Nagłówek X-OTCv8-API: 1.x (WS handshake).
* Zmiany łamiące → podbij **major** i utrzymuj deprecated min. 1 wersję.

## 8. Przykłady end‑to

‑

e

n

d

### 8.1. Status

d

o

U

I

-- modules/status/init.lua  
local M = {}  
function M.start(ctx)  
 ctx.emit("status", { text = "Ready" })  
end  
return M

### 8.2. Telemetria do dashbo

a

r

d

u

local M = {}  
function M.tick(ctx, dt)  
 ctx.ws.send("metrics.update", { hp = getHp(), mp = getMp() })  
end  
return M

### 8.3. Komenda z dashbo

a

r

d

u

**UI → WS**

{ "type": "cmd", "id": "run1", "name": "START", "args": { "profile": "pvp" } }

**WS → UI**

{ "type": "cmd.result", "id": "run1", "ok": true, "ts": 1712345678 }

## 9. Generowanie referencji z kodu (autom

a

t

y

)

### 9.1. Lua –

L

D

o

c

Komentuj funkcje:

--- Wysyła wiadomość do czatu.  
-- @tparam string msg  
-- @treturn boolean ok  
function say(msg) ... end

**Workflow**: ldoc -d docs/lua/ref/ . → włącz w mkdocs.yml (nav).

### 9.2. C++ – Dox

y

g

e

n

Doxyfile → HTML do docs/cpp/ref/ → dodaj do nav.

### 9.3. WS – JSON Schema → mark

d

o

w

n

Trzymaj schematy w schemas/ws/\*.schema.json i buduj referencję:

# przykład narzędz  
i  
  
a  
  
npx @apitable/json-schema-to-markdown schemas/ws -o docs/ws/ref.md

## 10. Lista TODO do uzupełni

e

n

i

a

* ☐ Pełna lista eventów Lua (nazwy + parametry)
* ☐ Mapowania ctx.emit(target) → UI (id + pola)
* ☐ Zamknięta lista ClientEvent/ServerEvent + schematy
* ☐ Nagłówki/entrypointy C++ i rejestracja hooków
* ☐ Wygeneruj LDoc/Doxygen i dodaj do nav