# 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

# 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

# Moduły (vBot)

# Moduły (vBot) — przewo

d

n

i

k

!!! info “Cel”

Jak tworzyć i ładować moduły Lua dla klienta OTCv8.

## Struktura mod

u

ł

u

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

## 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

## Rejestrowanie zdarzeń (przykł

a

d

)

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

## Konfigura

c

j

a

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

## Debug / l

o

g

i

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

## Dobre prakt

y

k

i

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

# Realtime (WebSocket)

# 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

## Zas

a

d

y

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

## Przykład (Node + socket

i

o

)

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

# OTUI

# OTUI — pods

t

a

w

y

!!! info

OTUI to deklaratywne layouty interfejsu użytkownika.

## Przykład layo

u

t

u

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

## Zdarzenia / wiąza

n

i

a

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

## Wskazó

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 (przykł

a

d

y

)

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

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

# Style guide

# 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.

# Przegląd

# C++ — prze

g

l

ą

d

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

# Narzędzia

# Narzę

d

z

i

a

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

# Contributing

# 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.

## St

y

l

e

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

## Commity dokumenta

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.

## Raport bł

ę

d

u

* Kroki odtworzenia, logi, wersja, OS.

# Release

# Rel

e

a

s

e

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

## Wersjonowanie dokumenta

c

j

i

* Docsy budują się z gałęzi master.

# Security

# 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](mailto:security@twojadomena.example)

# Troubleshooting

# Troubleshoo

t

i

n

g

## Bu

i

l

d

* Brak zależności vcpkg → zainstaluj pakiety i przebuduj cache.

## Andr

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

# Sło

w

n

i

k

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

# Roadmap

# Roa

d

m

a

p

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