# OTCv8 Dev Docs

# Start

# OTCv8 Dev — Dokumentacja

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 (skrót)

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) — przewodnik

!!! info “Cel” Jak tworzyć i ładować moduły Lua dla klienta OTCv8.

## Struktura modułu

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

## Minimalny moduł (Lua)

-- 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 (przykład)

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

## Konfiguracja

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

## Debug / logi

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

## Dobre praktyki

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

# Realtime (WebSocket)

# Realtime (WebSocket)

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

## Zasady

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

## Przykład (Node + socket.io)

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

# OTUI

# OTUI — podstawy

!!! info OTUI to deklaratywne layouty interfejsu użytkownika.

## Przykład layoutu

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

## Zdarzenia / wiązania

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

## Wskazówki

* 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 (przykłady)

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

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

# Style guide

# Lua — style guide

* 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++ — przegląd

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

# Narzędzia

# Narzędzia

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

# Contributing

# Contributing

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.

## Style

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

## Commity dokumentacji

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

# Testy

# Testy

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

## Raport błędu

* Kroki odtworzenia, logi, wersja, OS.

# Release

# Release

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

## Wersjonowanie dokumentacji

* Docsy budują się z gałęzi master.

# Security

# Security

* 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

# Troubleshooting

# Troubleshooting

## Build

* Brak zależności vcpkg → zainstaluj pakiety i przebuduj cache.

## Android

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

## WS

* Rozłączenia: sprawdź pingTimeout/pingInterval oraz proxy.

# FAQ

# FAQ

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

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

# Słownik

# Słownik

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

# Roadmap

# Roadmap

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