5.2 Описание библиотек

При разработке программного обеспечения целесообразно использовать готовый программный код.

Boost [23] – набор библиотек, использующих функциональность языка C++ и предоставляющих удобный кроссплатформенный высокоуровневый интерфейс для лаконичного кодирования различных повседневных подзадач программирования.

В программе используется библиотека asio из набора boost. Данная библиотека представляет функции асинхронной постановки задач, работы с сетевыми устройствами.

FMT [24] – библиотека для форматирования строк, предоставляющая удобный интерфейс при разработке программы. Используется для оформления отладочных сообщений, генерации токенов идемпотентности.

Json [25] – библиотека для работы с файлами JSON. Используется для работы с файлами конфигурации.

5.3 Настройка системы сборки

Для того что бы собрать проект средствами CMake, необходимо в корне дерева исходников разместить файл CMakeLists.txt, хранящий правила и цели сборки.

Сначала необходимо задать версию файла конфигурации, каталог с исходным кодом программы. Конфигурация проекта представлена на рисунке 5.1.

CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.14)
2   project(one-way-sync VERSION 0.1.0)
3
4   set(CMAKE_CXX_STANDARD 20)
5
6   include(FetchContent)
7
8   set (source_dir "${PROJECT_SOURCE_DIR}/src/")
9
10  file (GLOB source_files
11      "${source_dir}/*.cpp"
12      "${source_dir}/config/type/*.cpp"
13      "${source_dir}/config/*.cpp"
14      "${source_dir}/helpers/*.cpp"
15      "${source_dir}/models/*.cpp"
16      "${source_dir}/modules/*.cpp"
17      "${source_dir}/networking/*.cpp"
18  )
19
20  include_directories(
21      "${source_dir}/"
22  )
23
```

Рисунок 5.1 – Конфигурация проекта

Далее необходимо сконфигурировать библиотеки: указать каталог с исходниками, используемый стандарт языка, название переменных окружения. Конфигурация библиотек представлена на рисунке 5.2.

CMakeLists.txt

```
#Boost
set(BOOST_ROOT "C:/CLI_STUFF/mingw64/boost_1_78_0/")
set(CMAKE_INCLUDE_PATH ${CMAKE_INCLUDE_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/")
set(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/lib/")

set(Boost_USE_STATIC_LIBS ON)
find_package(Boost COMPONENTS system)

include_directories(${Boost_INCLUDE_DIR})

#CryptoPP
set(CryptoPP-header "C:/CLI_STUFF/mingw64/cryptopp860/")
set(CryptoPP-src "C:/CLI_STUFF/mingw64/cryptopp860/cryptopp/")

add_library(CryptoPP STATIC ${CryptoPP-src})
set_target_properties(CryptoPP PROPERTIES LINKER_LANGUAGE CXX)

include_directories(${CryptoPP-header})

#Threads
find_package(Threads REQUIRED)

#FMT
FetchContent_Declare(fmt
  GIT_REPOSITORY https://github.com/fmtlib/fmt.git
  GIT_TAG master
)
FetchContent_MakeAvailable(fmt)

#JSON-parsing
FetchContent_Declare(nlohmann_json
  GIT_REPOSITORY https://github.com/nlohmann/json
  GIT_TAG master
)
FetchContent_MakeAvailable(nlohmann_json)
```

Рисунок 5.2 – Конфигурация библиотек

Далее конфигурируются флаги компиляции, параметры выходного файла, методика подключения библиотек к итоговой программе. Конфигурация компилятора представлена на рисунке 5.3.

CMakeLists.txt

```
add_compile_options(-Wall -Wextra -pedantic -Werror -pthread)

add_executable(one-way-sync ${source_files})

target_link_libraries(one-way-sync ws2_32)
target_link_libraries(one-way-sync Threads::Threads)
target_link_libraries(one-way-sync CryptoPP)
target_link_libraries(one-way-sync fmt::fmt-header-only)
target_link_libraries(one-way-sync nlohmann_json::nlohmann_json)

```

Рисунок 5.3 – Конфигурация компилятора