Приложение Б
(обязательное)
Листинг исходного кода

Б.1 Текст программы «src\networking\udp_receiver.hpp»

```
#pragma once

#pragma once

#include <boost/asio.hpp>
#include <boost/asio/ip/address.hpp>
#include <boost/asio/ip/udp.hpp>
#include <config/type/udp_receiver.hpp>
#include <functional>
#include <models/udp_buffer.hpp>
#include <optional>
#include <vector>

namespace networking::udp_receiver {

namespace ip = boost::asio::ip;

class UdpReceiver {
 public:
  using HandlerType = std::function<void(models::udp_buffer::DataBuffer&)>;

  UdpReceiver(const config::udp_receiver::Configuration& config,
              const HandlerType handler)
      : started_(false),
        thread_count_(config.receive_threads),
        ip_version_(config.ip_version),
        buffer_queue_(config.buffer_count, config.datagram_max_size),
        socket_(io_service_),
        receiver_endpoint_(boost::asio::ip::address_v4::any(),
                           config.target_port),
        expected_source_ip_(config.target_ip),
        handler_(handler){};

  void Start();

  class ReceiveFailed : std::runtime_error {
    using std::runtime_error::runtime_error;
  };
  class BufferCorrupted : std::runtime_error {
    using std::runtime_error::runtime_error;
  };

  class BufferSizeLessThanRequired : std::runtime_error {
    using std::runtime_error::runtime_error;
  };

  ~UdpReceiver() {
    io_service_.stop();
    socket_.close();

    for (auto& t : receive_threads_) {
      //End threads and supress exceptions if present;
      try {
        t.join();
      } catch (const std::exception&) {}
    }
  }

 private:
  void WaitReceive();
  void Handle(models::udp_buffer::DataBuffer& data_buffer,
              ip::udp::endpoint& udp_source,
              const boost::system::error_code& error, size_t bytes_transferred);

  bool started_;
  int thread_count_;
  ip::udp ip_version_;
  models::udp_buffer::BufferQueue buffer_queue_;
  boost::asio::io_service io_service_;
  ip::udp::socket socket_{io_service_};
  boost::asio::io_service::work work_{io_service_};
  ip::udp::endpoint receiver_endpoint_;
  std::optional<ip::address> expected_source_ip_;
  std::vector<std::thread> receive_threads_;
  HandlerType handler_;
};

}  // namespace networking::udp_receiver
```

Б.2 Текст программы «src\networking\udp_receiver.cpp»

```
#include "udp_receiver.hpp"

#include <fmt/core.h>
```

```
#include <boost/bind/bind.hpp>
#include <unordered_set>
#include <utils/logging.hpp>

namespace networking::udp_receiver {
namespace {
struct RAIIBufferContainer {
  RAIIBufferContainer(
      std::unique_ptr<models::udp_buffer::DataBuffer> buffer_to_return,
      models::udp_buffer::BufferQueue& queue_to_return)
      : container(std::move(buffer_to_return)), queue(queue_to_return) {}
  ~RAIIBufferContainer() { queue.ReleaseBuffer(std::move(container)); }
  std::unique_ptr<models::udp_buffer::DataBuffer> container;
  models::udp_buffer::BufferQueue& queue;
};

}  // namespace

void UdpReceiver::Start() {
  if (started_) {
    LOG_INFO() << "UdpReceiver already started\n";
    return;
  }
  started_ = true;

  socket_.open(ip_version_);
  socket_.bind(receiver_endpoint_);

  for (int i = 0; i < thread_count_; i++) {
    receive_threads_.push_back(std::thread([this] {
      WaitReceive();
      io_service_.run();
    }));
  }
}

void UdpReceiver::WaitReceive() {
  auto buffer = buffer_queue_.AquireBuffer();
  auto endpoint = std::make_unique<ip::udp::endpoint>();

  socket_.async_receive_from(
      boost::asio::buffer(buffer->buffer, buffer->buffer.size()), *endpoint,
      [data_buffer = std::move(buffer), udp_source = std::move(endpoint), this](
          const boost::system::error_code& error,
          size_t bytes_transferred) mutable {
```

```
        if (!data_buffer) {
          throw UdpReceiver::BufferCorrupted("Data buffer ptr is NULL");
        }
        if (!udp_source) {
          throw UdpReceiver::BufferCorrupted("UdpSource lost");
        }

        RAIIBufferContainer secured_buffer(std::move(data_buffer),
                                           buffer_queue_);

        Handle(*secured_buffer.container, *udp_source, error,
               bytes_transferred);
      });
}

void UdpReceiver::Handle(models::udp_buffer::DataBuffer& data_buffer,
                         ip::udp::endpoint& udp_source,
                         const boost::system::error_code& error,
                         size_t bytes_transferred) {
  static const std::unordered_set<int> ignored_errors{
      boost::asio::error::operation_aborted};

  if (error) {
    if (!ignored_errors.contains(error.value())) {
      LOG_WARNING() << fmt::format("Receive failed: {} | Val: {} | Cat: {}\n",
                                   error.message(), error.value(),
                                   error.category().name());
    }
    return;
  }

  if (expected_source_ip_) {
    if (expected_source_ip_ != udp_source.address()) {
      LOG_INFO() << "Receive blocked for ip: " << udp_source.address() << "\n";
      return;
    }
  }

  if (data_buffer.buffer.size() < bytes_transferred) {
    throw UdpReceiver::BufferSizeLessThanRequired("Data buffer ptr is NULL");
  }
  data_buffer.last_datagram_size = bytes_transferred;

  try {
    handler_(data_buffer);
  } catch (std::runtime_error& e) {
    LOG_INFO() << "Caught untyped exception: " << e.what() << "\n";
  }
}

}  // namespace networking::udp_receiver
```

Б.3 Текст программы «src\networking\udp_sender.hpp»

```
#pragma once

#include <boost/asio.hpp>
#include <boost/asio/ip/address.hpp>
#include <boost/asio/ip/udp.hpp>
#include <config/type/udp_sender.hpp>
#include <vector>

namespace networking::udp_sender {

namespace ip = boost::asio::ip;

class UdpSender {
 public:
  struct Package {
    std::vector<char> data;
    int retries;
  };

  class DataIncorrectState : std::runtime_error {
    using std::runtime_error::runtime_error;
  };

  UdpSender(const config::udp_sender::Configuration& config)
      : receiver_endpoint_(config.target_ip, config.target_port),
        retry_count_(config.retry_count) {
    socket_.open(config.ip_version);
  }

  void Start();

  void Send(std::unique_ptr<Package> data_to_send);

  ~UdpSender() {
    io_service_.stop();
    socket_.close();

    try {
      sender_thread_->join();
    } catch (const std::exception&) {
    }
  }

 private:
  ip::udp::endpoint receiver_endpoint_;
  boost::asio::io_service io_service_;
  boost::asio::io_service::work work_{io_service_};
  ip::udp::socket socket_{io_service_};
  std::unique_ptr<std::thread> sender_thread_;
  int retry_count_;
};


}  // namespace networking::udp_sender
```

Б.4 Текст программы «src\networking\udp_sender.cpp»

```
#include "udp_sender.hpp"

#include <fmt/core.h>

#include <utils/logging.hpp>

namespace networking::udp_sender {

void UdpSender::Start() {
  sender_thread_ =
      std::make_unique<std::thread>(std::thread([this] { io_service_.run(); }));
}

void UdpSender::Send(std::unique_ptr<UdpSender::Package> data_to_send) {
  if (!data_to_send) {
    throw DataIncorrectState("Data package handles nothing");
  }

  socket_.async_send_to(
      boost::asio::buffer(data_to_send->data), receiver_endpoint_,
      [data_buffer = std::move(data_to_send), this](
          const boost::system::error_code& error,
          size_t bytes_transferred) mutable {
        if (!error && data_buffer->data.size() == bytes_transferred) {
          /*Send succeed*/
          return;
        }
        LOG_WARNING() << fmt::format(
            "Send failed: {} \ndata.size()::{} / actually_sent::{}\n",
            error.what(), data_buffer->data.size(), bytes_transferred);

        if (data_buffer->retries > retry_count_) {
          LOG_WARNING() << "Data sent failed after retries: "
                        << data_buffer->retries;
          return;
        }

        data_buffer->retries += 1;
        Send(std::move(data_buffer));
      });
}

}  // namespace networking::udp_sender
```

Б.5 Текст программы «src\modules\fs_watcher.hpp»

```
#pragma once

#include <boost/asio.hpp>
#include <config/type/fs_watcher.hpp>
#include <filesystem>
#include <functional>
#include <optional>
#include <thread>
#include <unordered_map>
#include <utils/call_each.hpp>

namespace modules::fs_watcher {

enum class FileAction { kCreated, kModified, kDeleted };

struct FileDiscription {
  std::filesystem::path path;
  uintmax_t size;
  std::filesystem::file_time_type last_write_time;

  bool operator==(const FileDiscription& other) const {
    return path == other.path && size == other.size &&
           last_write_time == other.last_write_time;
  }
};
class FileWatcher {
 public:
  using FileUpdateHandler =
      std::function<void(const FileDiscription&, FileAction)>;

  FileWatcher(config::filesystem_watcher::Configuration& config,
              std::optional<FileUpdateHandler> handler)
      : folder_path_(config.folder_path), handler_(handler) {
    namespace fs = std::filesystem;

    for (auto& file : fs::recursive_directory_iterator(folder_path_)) {
      if (fs::is_regular_file(file)) {
        cache_[file.path().string()] =
            std::make_unique<FileDiscription>(FileDiscription{
                file.path(), file.file_size(), file.last_write_time()});
      }
    }

    if (handler) {
      caller_ = std::make_unique<utils::call_each::CallEach>(
          [this]() { UpdateFolder(); }, config.check_folder_each);
    }
  }

  void UpdateFolder();

  void List(std::function<void(const FileDiscription&)> info_cb);

 private:
  using FileWatcherCache =
      std::unordered_map<std::string, std::unique_ptr<FileDiscription>>;
  std::string folder_path_;
  std::optional<FileUpdateHandler> handler_;
  std::unique_ptr<utils::call_each::CallEach> caller_;
  std::mutex mutex_;
  FileWatcherCache cache_;
};

}  // namespace modules::filesystem_watcher
```

Б.6 Текст программы «src\modules\fs_watcher.cpp»

```
 #include "fs_watcher.hpp"

namespace modules::fs_watcher {
```

```
void FileWatcher::UpdateFolder() {
  namespace fs = std::filesystem;

  const std::lock_guard<std::mutex> lock(mutex_);

  FileWatcherCache prev_cache_state_;
  cache_.swap(prev_cache_state_);

  for (auto& file :
       std::filesystem::recursive_directory_iterator(folder_path_)) {
    if (!fs::is_regular_file(file)) {
      continue;
    }
    auto current_file_last_write_time = std::filesystem::last_write_time(file);
    auto file_size = file.file_size();

    auto prev_file_state = prev_cache_state_.find(file.path().string());

    auto file_discription =
        FileDiscription{file.path(), file_size, current_file_last_write_time};

    if (prev_file_state == prev_cache_state_.end()) {
      // New file
      if (handler_) {
        handler_.value()(file_discription, FileAction::kCreated);
      }
      cache_[file.path().string()] =
          std::make_unique<FileDiscription>(std::move(file_discription));

    } else {
      // Exists
      if (!(*prev_file_state->second == file_discription)) {
        // Has changes
        if (handler_) {
          handler_.value()(file_discription, FileAction::kModified);
        }
      }
      prev_cache_state_.erase(prev_file_state);
    }

    cache_[file.path().string()] =
        std::make_unique<FileDiscription>(std::move(file_discription));
  }

  for(const auto& [path, description] : prev_cache_state_){
    if (handler_) {
      handler_.value()(*description, FileAction::kDeleted);
    }
  }
}

void FileWatcher::List(std::function<void(const FileDiscription&)> info_cb){
  const std::lock_guard<std::mutex> lock(mutex_);

  for(const auto& [path, description] : cache_){
    info_cb(*description);
  }
}

}  // namespace modules::filesystem_watcher
```

Б.7 Текст программы «src\modules\asio_job_queue.hpp»

```
#include <boost/asio.hpp>
#include <boost/bind/bind.hpp>
#include <functional>
#include <iostream>
#include <thread>

namespace modules::asio_job_queue {

class AsioJobQueue {
 public:
  AsioJobQueue(std::size_t pool_size) : work_(io_service_) {
    for (std::size_t i = 0; i < pool_size; ++i) {
      threads_.push_back(std::thread([this] { io_service_.run(); }));
    }
  }

  ~AsioJobQueue(){
    io_service_.stop();
    for (auto& t : threads_) {
      //End threads and supress exceptions if present;
      try {
        t.join();
      } catch (const std::exception&) {}
    }
  }

  template <typename Task>
  void Shedule(Task task) {
    const std::lock_guard<std::mutex> lock(mutex_);

    io_service_.post([this, &task]() { Wrapper(std::function<void()>(task)); });
  }

 private:
  void Wrapper(std::function<void()> task);

  boost::asio::io_service io_service_;
  boost::asio::io_service::work work_;
  std::vector<std::thread> threads_;
  std::mutex mutex_;
};

}  // namespace modules::asio_job_queue
```

Б.8 Текст программы «src\modules\asio_job_queue.cpp»

```
#include "asio_job_queue.hpp"

namespace modules::asio_job_queue {

void AsioJobQueue::Wrapper(std::function<void()> task) {
  try {
    task();
  } catch (const std::exception& e) {
    std::cout << "Exception caught in wrapper" << e.what() << "\n";
  } catch (const std::runtime_error& e) {
    std::cout << "Exception caught in wrapper" << e.what() << "\n";
  }
}

}  // namespace modules::asio_job_queue
```

Б.9 Текст программы «src\modules\file_scheduler.hpp»

```
#pragma once

#include <fstream>
#include <functional>
#include <models/file_frame.hpp>
#include <modules/fs_watcher.hpp>

namespace helpers::file_scheduler {
```

```
class FileReader {
 public:
  using ReadCallback = std::function<void(models::file_frame::FileFragment&&)>;
  FileReader(const modules::fs_watcher::FileDiscription& discription,
             uint64_t max_frame_size)
      : max_frame_size_(max_frame_size),
        file_(discription_.path, std::ios::binary),
        discription_(discription) {}

  void Read(ReadCallback callback, models::file_frame::Action action);
  bool OpenOk();

  ~FileReader() { file_.close(); }

 private:
  uint64_t max_frame_size_;
  std::ifstream file_;
  modules::fs_watcher::FileDiscription discription_;
};

}  // namespace helpers::file_scheduler
```

Б.10 Текст программы «src\models\file_frame.hpp»

```
#pragma once

#include <filesystem>

#include <string>
#include <vector>

#include <helpers/serialization.hpp>
#include <helpers/to.hpp>

namespace models::file_frame {

using Payload = std::vector<char>;

enum class Action {
  kCreate,
  kModify,
  kDelete
};

std::string ToString(const Action&);
Action Convert(const std::string&, helpers::to::To<Action>);

struct FileHeader {
  std::string path;
  uintmax_t size;
  Action action;
  std::filesystem::file_time_type last_write_time;
  std::string sha256_hash;
};

struct FragmentHeader {
  uint64_t part;
  uint64_t parts;
  std::string part_sha256_hash;
  uint64_t write_position;
};

struct FileFragment {
  FileHeader file_header;
  FragmentHeader fragment_header;
  std::string idempotency_token;
  Payload payload;
};

}  // namespace models::file_frame

namespace helpers::serialization {

SerializedContainer Serialize(
    const models::file_frame::FileFragment& to_serialize);

models::file_frame::FileFragment Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<models::file_frame::FileFragment>);

}  // namespace helpers::serialization
```

Б.11 Текст программы «src\models\file_frame.cpp»

```
#include "file_frame.hpp"

#include <unordered_map>

#include <models/idempotency_token.hpp>

namespace models::file_frame {
```

```
std::string ToString(const Action& action) {
  switch (action) {
    case Action::kCreate:
      return "create";
    case Action::kModify:
      return "modify";
    case Action::kDelete:
      return "delete";
  }
}
Action Convert(const std::string& from, helpers::to::To<Action>) {
  static const std::unordered_map<std::string, Action> convert{
      {"create", Action::kCreate},
      {"modify", Action::kModify},
      {"delete", Action::kDelete}};
  auto find = convert.find(from);
  if (find == convert.end()) {
    throw helpers::to::ConvertError(std::string("Can't convert Action from: ") +
                                    from);
  }
  return find->second;
}

}  // namespace models::file_frame

namespace helpers::serialization {

SerializedContainer Serialize(
    const models::file_frame::FileHeader& to_serialize) {
  SerializedContainer out;

  auto write_time = to_serialize.last_write_time.time_since_epoch().count();

  auto path_part = Serialize(to_serialize.path);
  auto size_part = Serialize(to_serialize.size);
  auto action_part = Serialize(ToString(to_serialize.action));
  auto last_write_part = Serialize(write_time);
  auto sha_part = Serialize(to_serialize.sha256_hash);

  out.reserve(path_part.size() + size_part.size() + action_part.size() +
              last_write_part.size() + action_part.size());

  out.insert(out.end(), path_part.begin(), path_part.end());
  out.insert(out.end(), size_part.begin(), size_part.end());
  out.insert(out.end(), action_part.begin(), action_part.end());
  out.insert(out.end(), last_write_part.begin(), last_write_part.end());
  out.insert(out.end(), sha_part.begin(), sha_part.end());

  return out;
}

models::file_frame::FileHeader Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<models::file_frame::FileHeader>){

  auto path = Deserialize(start_position, to::To<std::string>());
  auto size = Deserialize(start_position, to::To<uintmax_t>());
  auto action_str = Deserialize(start_position, to::To<std::string>());
  auto last_write_epoch = Deserialize(start_position, to::To<int64_t>());
  auto sha256 = Deserialize(start_position, to::To<std::string>());

  std::chrono::time_point<std::chrono::system_clock, std::chrono::milliseconds>
    last_write_tp{std::chrono::milliseconds{last_write_epoch}};

  return models::file_frame::FileHeader{
    path,
    size,
    Convert(action_str, helpers::to::To<models::file_frame::Action>()),
    std::chrono::file_clock::from_sys(last_write_tp),
    sha256
  };
}

SerializedContainer Serialize(
    const models::file_frame::FragmentHeader& to_serialize) {
  SerializedContainer out;

  auto part = Serialize(to_serialize.part);
  auto parts = Serialize(to_serialize.parts);
  auto part_sha256_hash = Serialize(to_serialize.part_sha256_hash);
  auto write_position = Serialize(to_serialize.write_position);

  out.reserve(part.size() + parts.size() + part_sha256_hash.size() +
            write_position.size());

  out.insert(out.end(), part.begin(), part.end());
  out.insert(out.end(), parts.begin(), parts.end());
  out.insert(out.end(), part_sha256_hash.begin(), part_sha256_hash.end());
  out.insert(out.end(), write_position.begin(), write_position.end());

  return out;
```

```
}

models::file_frame::FragmentHeader Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<models::file_frame::FragmentHeader>){

  auto part = Deserialize(start_position, to::To<uint64_t>());
  auto parts = Deserialize(start_position, to::To<uint64_t>());
  auto part_sha256_hash = Deserialize(start_position, to::To<std::string>());
  auto write_position = Deserialize(start_position, to::To<uint64_t>());

  return models::file_frame::FragmentHeader{
    part,
    parts,
    part_sha256_hash,
    write_position
  };
}

SerializedContainer Serialize(
    const models::file_frame::FileFragment& to_serialize) {
  SerializedContainer out;

  auto file_header = Serialize(to_serialize.file_header);
  auto fragment_header = Serialize(to_serialize.fragment_header);
  auto idempotency_token = Serialize(to_serialize.idempotency_token);
  auto payload = Serialize(to_serialize.payload);

  out.reserve(file_header.size() + fragment_header.size() + idempotency_token.size() +
payload.size());

  out.insert(out.end(), file_header.begin(), file_header.end());
  out.insert(out.end(), fragment_header.begin(), fragment_header.end());
  out.insert(out.end(), idempotency_token.begin(), idempotency_token.end());
  out.insert(out.end(), payload.begin(), payload.end());

  return out;
}


models::file_frame::FileFragment Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<models::file_frame::FileFragment>){

  auto file_header = Deserialize(start_position, to::To<models::file_frame::FileHeader>());
  auto fragment_header = Deserialize(start_position, to::To<models::file_frame::FragmentHeader>());
  auto idempotency_token = Deserialize(start_position, to::To<std::string>());
  auto payload = Deserialize(start_position, to::To<models::file_frame::Payload>());

  return models::file_frame::FileFragment{
    file_header,
    fragment_header,
    idempotency_token,
    payload
  };
}

}  // namespace helpers::serialization
```

Б.12 Текст программы «src\models\idempotency_token.hpp»

```
#pragma once

#include <string>

#include <models/file_frame.hpp>

namespace models::idempotency_token {

std::string GetToken(const file_frame::FileHeader& file_header,
              const file_frame::FragmentHeader& fragment_header);

}
```

Б.13 Текст программы «src\models\idempotency_token.cpp»

```
#include "idempotency_token.hpp"

#include <fmt/core.h>

namespace models::idempotency_token {

std::string GetToken(const file_frame::FileHeader& file_header,
                    const file_frame::FragmentHeader& fragment_header) {
  return fmt::format("{}-{}-{}-{}-{}", file_header.path,
                    file_header.last_write_time.time_since_epoch().count(),
                    file_header.size, fragment_header.part,
                    fragment_header.parts);
}
}  // namespace models::idempotency_token
```

Б.14 Текст программы «src\models\udp_buffer.hpp»

```
#pragma once

#include <atomic>
#include <exception>
#include <memory>
#include <mutex>
#include <vector>

namespace models::udp_buffer {

struct DataBuffer {
  DataBuffer(int _last_datagram_size, std::vector<char>&& _buffer)
      : last_datagram_size(_last_datagram_size), buffer(_buffer) {}
  int last_datagram_size;
  std::vector<char> buffer;
};

class BufferQueue {
 public:
  class AllBuffersLocked : std::runtime_error {
    using std::runtime_error::runtime_error;
  };

  BufferQueue(int queue_size, int buffer_size);

  std::unique_ptr<DataBuffer> AquireBuffer();
  void ReleaseBuffer(std::unique_ptr<DataBuffer> buffer_to_release);

 private:
  int buffer_size_;
  std::vector<std::unique_ptr<DataBuffer>> free_buffers_;
  std::mutex mutex_;
};

}  // namespace models::udp_buffer
```

Б.15 Текст программы «src\models\udp_buffer.cpp»

```
#include "udp_buffer.hpp"
```

```
#include <utils/logging.hpp>

namespace models::udp_buffer {


BufferQueue::BufferQueue(int queue_size, int buffer_size) {
  buffer_size_ = buffer_size;

  free_buffers_.reserve(queue_size);

  for (int i = 0; i < queue_size; i++) {
    free_buffers_.emplace_back(
      std::make_unique<DataBuffer>(0, std::vector<char>(buffer_size_))
    );
  }
}

std::unique_ptr<DataBuffer> BufferQueue::AquireBuffer() {
  std::lock_guard lock(mutex_);

  if(free_buffers_.size() == 0){
    buffer_size_+=1;
    LOG_INFO() << "Added extra buffer, now total: " << buffer_size_ << '\n';

    free_buffers_.emplace_back(
      std::make_unique<DataBuffer>(0, std::vector<char>(buffer_size_))
    );
  }

  auto buffer = std::move(free_buffers_.back());
  free_buffers_.pop_back();

  return buffer;
}

void BufferQueue::ReleaseBuffer(std::unique_ptr<DataBuffer> buffer_to_release){
  std::lock_guard lock(mutex_);

  free_buffers_.push_back(std::move(buffer_to_release));
}

}  // namespace models::udp_buffer
```

Б.16 Текст программы «src\models\udp_frame_queue.hpp»

```
#pragma once
```

```
#include <boost/asio.hpp>
#include <models/udp_buffer.hpp>
#include <modules/asio_job_queue.hpp>
#include <vector>

namespace networking::frame_queue {

namespace ip = boost::asio::ip;

class FrameQueue {
public:
  FrameQueue(int workers_count) : strand_(io_service_), pool_(workers_count) {}

  void Add(models::udp_buffer::DataBuffer& frame_buffer);

private:
  boost::asio::io_service io_service_;
  boost::asio::io_service::strand strand_;
  modules::asio_job_queue::AsioJobQueue pool_;
  std::mutex mutex_;
};

}  // namespace networking::frame_queue
```

Б.17 Текст программы «src\models\udp_frame_queue.cpp»

```
#include "udp_frame_queue.hpp"

#include <fmt/core.h>

namespace networking::frame_queue {

void FrameQueue::Add(models::udp_buffer::DataBuffer& frame_buffer) {
  // Perform pre_process checks
  // Structural_binding
  pool_.Shedule([&frame_buffer]() {
    std::cout << "FrameQueue" << frame_buffer.last_datagram_size;
    std::cout << fmt::format("Processed Frame with size: {}\n",
                             frame_buffer.last_datagram_size);
  });
}

}  // namespace networking::frame_queue
```

Б.18 Текст программы «src\helpers\serialization.hpp»

```
#pragma once

#include <helpers/to.hpp>
#include <string>
#include <tuple>
#include <vector>

namespace helpers::serialization {

static const int kSizeBytes = 8;
using SerializedContainer = std::vector<char>;

SerializedContainer Serialize(const int64_t& to_serialize);
int64_t Deserialize(SerializedContainer::const_iterator& start_position, to::To<int64_t>);

SerializedContainer Serialize(const uint64_t& to_serialize);
uint64_t Deserialize(SerializedContainer::const_iterator& start_position,
                     to::To<uint64_t>);

SerializedContainer Serialize(const std::string& to_serialize);
std::string Deserialize(SerializedContainer::const_iterator& start_position,
                        to::To<std::string>);

SerializedContainer Serialize(const std::vector<char>& to_serialize);
std::vector<char> Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<std::vector<char>>);

}  // namespace helpers::serialization
```

Б.19 Текст программы «src\helpers\serialization.cpp»

```
#include "serialization.hpp"

namespace helpers::serialization {

namespace {

template<typename IntType>
SerializedContainer SerializeInt(const IntType& to_serialize) {
  SerializedContainer out;
  out.reserve(sizeof(IntType));
  for (int offset = sizeof(IntType) * 8 - 8; offset >= 0; offset -= 8) {
    out.push_back(to_serialize & (0xFFULL << (offset)));
  }
  return out;
}

template<typename IntType>
IntType DeserializeInt(SerializedContainer::const_iterator& start_position) {
  IntType out = 0;

  const auto end = start_position + sizeof(IntType);

  for (auto it = start_position; it != end; it++) {
    out = (out << 8) | *it;
  }

  start_position = end;
  return out;
}

template<typename Container>
SerializedContainer SerializeContainer(const Container& to_serialize) {
  SerializedContainer container;
  container.reserve(to_serialize.size() + kSizeBytes);

  auto size_part = Serialize(to_serialize.size());
  container.insert(container.begin(), size_part.begin(), size_part.end());
  container.insert(container.end(), to_serialize.begin(), to_serialize.end());

  return container;
}

template<typename Container>
Container DeserializeContainer(SerializedContainer::const_iterator& start_position){
  const auto size = Deserialize(start_position, to::To<std::uint64_t>());
  const auto string_start = start_position + kSizeBytes;

  start_position = string_start + size;

  Container out(string_start, string_start + size);
  return out;
}
}

SerializedContainer Serialize(const int64_t& to_serialize) {
  return SerializeInt<int64_t>(to_serialize);
```

```
}

int64_t Deserialize(SerializedContainer::const_iterator& start_position,
                    to::To<int64_t>) {
  return DeserializeInt<int64_t>(start_position);
}

SerializedContainer Serialize(const uint64_t& to_serialize) {
  return SerializeInt<uint64_t>(to_serialize);
}

uint64_t Deserialize(SerializedContainer::const_iterator& start_position,
                    to::To<uint64_t>) {
  return DeserializeInt<uint64_t>(start_position);
}

SerializedContainer Serialize(const std::string& to_serialize) {
  return SerializeContainer<std::string>(to_serialize);
}
std::string Deserialize(SerializedContainer::const_iterator& start_position,
                        to::To<std::string>){
  return DeserializeContainer<std::string>(start_position);
}

SerializedContainer Serialize(const std::vector<char>& to_serialize) {
  return SerializeContainer<std::vector<char>>(to_serialize);
}
std::vector<char> Deserialize(SerializedContainer::const_iterator& start_position,
                        to::To<std::vector<char>>){
  return DeserializeContainer<std::vector<char>>(start_position);
}

}
```

Б.20 Текст программы «src\helpers\to.hpp»

```
#pragma once

#include <stdexcept>

namespace helpers::to {

template <typename T>
struct To {};

class ConvertError : std::runtime_error {
  using std::runtime_error::runtime_error;
};

class SerializationError : std::runtime_error {
  using std::runtime_error::runtime_error;
};

}  // namespace helpers::to
```

Б.21 Текст программы «src\utils\calculate_hash.hpp»

```
#pragma once

#include <string>
#include <vector>
#include <fstream>

namespace utils::calculate_hash{

std::string Calculate(std::ifstream& file);
std::string Calculate(const std::vector<char>& data);

} // namespace utils::calculate_hash
```

Б.22 Текст программы «src\utils\calculate_hash.cpp»

```
#include "calculate_hash.hpp"

#include <utils/sha256.h>

namespace utils::calculate_hash {

std::string Calculate(std::ifstream& file) {
  return sha256(file);
}

std::string Calculate(const std::vector<char>& data) {
  return sha256(data);
}

}  // namespace utils::calculate_hash
```

Б.23 Текст программы «src\utils\call_each.hpp»

```
#pragma once

#include <boost/asio.hpp>
#include <functional>

namespace utils::call_each {

class CallEach {
 public:
  using Handler = std::function<void()>;
  CallEach(Handler handler, std::chrono::seconds delay) : handler_(handler),
      check_each_(delay), thread_([this] {
        Update();
        io_service_.run();
      }) {}

  ~CallEach() {
    io_service_.stop();
    try {
      thread_.join();
    } catch (const std::exception&) {
    }
  }

 private:
  void Update() {
    timer_.expires_from_now(check_each_);
    timer_.async_wait([this](const boost::system::error_code& e) {
      if(!e || e.value() != boost::asio::error::operation_aborted){
        handler_();
      }
      Update();
    });
  }
  Handler handler_;
  boost::asio::io_service io_service_;
  boost::asio::io_service::work work_{io_service_};
  boost::asio::steady_timer timer_{io_service_};
  std::chrono::seconds check_each_;
  std::thread thread_;
};
}  // namespace utils::call_each
```

Б.24 Текст программы «src\utils\logging.hpp»

```
#pragma once

#include <iostream>

#define LOG_TARGET std::cout

#define _LOG_DEBUG true
#define LOG_DEBUG() if(_LOG_DEBUG) LOG_TARGET

#define _LOG_INFO true
#define LOG_INFO() if(_LOG_INFO) LOG_TARGET

#define _LOG_WARNING true
#define LOG_WARNING() if(_LOG_WARNING) LOG_TARGET
```

Б.25 Текст программы «src\utils\sha256.h»

```
#pragma once

#include <string>
#include <stdint.h>

namespace utils::sha256 {

class SHA256
{
public:
  enum { BlockSize = 512 / 8, HashBytes = 32 };

  SHA256();

  std::string operator()(const void* data, size_t numBytes);
  std::string operator()(const std::string& text);

  void add(const void* data, size_t numBytes);

  std::string getHash();
  void        getHash(unsigned char buffer[HashBytes]);

  void reset();

private:
  void processBlock(const void* data);
  void processBuffer();

  uint64_t m_numBytes;
  size_t   m_bufferSize;
  uint8_t  m_buffer[BlockSize];

  enum { HashValues = HashBytes / 4 };
  uint32_t m_hash[HashValues];
};

}
```

Б.26 Текст программы «src\utils\sha256.cpp»

```
#include "sha256.h"

namespace utils::sha256 {
/// same as reset()
SHA256::SHA256()
{
  reset();
}


/// restart
void SHA256::reset()
{
  m_numBytes   = 0;
  m_bufferSize = 0;

  // according to RFC 1321
  m_hash[0] = 0x6a09e667;
  m_hash[1] = 0xbb67ae85;
  m_hash[2] = 0x3c6ef372;
  m_hash[3] = 0xa54ff53a;
  m_hash[4] = 0x510e527f;
  m_hash[5] = 0x9b05688c;
  m_hash[6] = 0x1f83d9ab;
  m_hash[7] = 0x5be0cd19;
}


namespace
{
  inline uint32_t rotate(uint32_t a, uint32_t c)
  {
    return (a >> c) | (a << (32 - c));
  }

  inline uint32_t swap(uint32_t x)
  {
#if defined(__GNUC__) || defined(__clang__)
    return __builtin_bswap32(x);
#endif
#ifdef MSC_VER
    return _byteswap_ulong(x);
#endif

    return (x >> 24) |
          ((x >>  8) & 0x0000FF00) |
          ((x <<  8) & 0x00FF0000) |
           (x << 24);
  }

  // mix functions for processBlock()
  inline uint32_t f1(uint32_t e, uint32_t f, uint32_t g)
  {
    uint32_t term1 = rotate(e, 6) ^ rotate(e, 11) ^ rotate(e, 25);
    uint32_t term2 = (e & f) ^ (~e & g); //(g ^ (e & (f ^ g)))
    return term1 + term2;
  }

  inline uint32_t f2(uint32_t a, uint32_t b, uint32_t c)
  {
    uint32_t term1 = rotate(a, 2) ^ rotate(a, 13) ^ rotate(a, 22);
    uint32_t term2 = ((a | b) & c) | (a & b); //(a & (b ^ c)) ^ (b & c);
    return term1 + term2;
  }
}


/// process 64 bytes
void SHA256::processBlock(const void* data)
{
  // get last hash
  uint32_t a = m_hash[0];
  uint32_t b = m_hash[1];
  uint32_t c = m_hash[2];
  uint32_t d = m_hash[3];
  uint32_t e = m_hash[4];
  uint32_t f = m_hash[5];
  uint32_t g = m_hash[6];
```

```
    uint32_t h = m_hash[7];

    // data represented as 16x 32-bit words
    const uint32_t* input = (uint32_t*) data;
    // convert to big endian
    uint32_t words[64];
    int i;
    for (i = 0; i < 16; i++)
  #if defined(__BYTE_ORDER) && (__BYTE_ORDER != 0) && (__BYTE_ORDER ==
__BIG_ENDIAN)
      words[i] =      input[i];
  #else
      words[i] = swap(input[i]);
  #endif

    uint32_t x,y; // temporaries

    // first round
    x = h + f1(e,f,g) + 0x428a2f98 + words[ 0]; y = f2(a,b,c); d += x; h = x + y;
    x = g + f1(d,e,f) + 0x71374491 + words[ 1]; y = f2(h,a,b); c += x; g = x + y;
    x = f + f1(c,d,e) + 0xb5c0fbcf + words[ 2]; y = f2(g,h,a); b += x; f = x + y;
    x = e + f1(b,c,d) + 0xe9b5dba5 + words[ 3]; y = f2(f,g,h); a += x; e = x + y;
    x = d + f1(a,b,c) + 0x3956c25b + words[ 4]; y = f2(e,f,g); h += x; d = x + y;
    x = c + f1(h,a,b) + 0x59f111f1 + words[ 5]; y = f2(d,e,f); g += x; c = x + y;
    x = b + f1(g,h,a) + 0x923f82a4 + words[ 6]; y = f2(c,d,e); f += x; b = x + y;
    x = a + f1(f,g,h) + 0xab1c5ed5 + words[ 7]; y = f2(b,c,d); e += x; a = x + y;

    // secound round
    x = h + f1(e,f,g) + 0xd807aa98 + words[ 8]; y = f2(a,b,c); d += x; h = x + y;
    x = g + f1(d,e,f) + 0x12835b01 + words[ 9]; y = f2(h,a,b); c += x; g = x + y;
    x = f + f1(c,d,e) + 0x243185be + words[10]; y = f2(g,h,a); b += x; f = x + y;
    x = e + f1(b,c,d) + 0x550c7dc3 + words[11]; y = f2(f,g,h); a += x; e = x + y;
    x = d + f1(a,b,c) + 0x72be5d74 + words[12]; y = f2(e,f,g); h += x; d = x + y;
    x = c + f1(h,a,b) + 0x80deb1fe + words[13]; y = f2(d,e,f); g += x; c = x + y;
    x = b + f1(g,h,a) + 0x9bdc06a7 + words[14]; y = f2(c,d,e); f += x; b = x + y;
    x = a + f1(f,g,h) + 0xc19bf174 + words[15]; y = f2(b,c,d); e += x; a = x + y;

    // extend to 24 words
    for (; i < 24; i++)
      words[i] = words[i-16] +
            (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
            words[i-7] +
            (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

    // third round
    x = h + f1(e,f,g) + 0xe49b69c1 + words[16]; y = f2(a,b,c); d += x; h = x + y;
```

```
x = g + f1(d,e,f) + 0xefbe4786 + words[17]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0x0fc19dc6 + words[18]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0x240ca1cc + words[19]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0x2de92c6f + words[20]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0x4a7484aa + words[21]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0x5cb0a9dc + words[22]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0x76f988da + words[23]; y = f2(b,c,d); e += x; a = x + y;

// extend to 32 words
for (; i < 32; i++)
  words[i] = words[i-16] +
             (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
             words[i-7] +
             (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// fourth round
x = h + f1(e,f,g) + 0x983e5152 + words[24]; y = f2(a,b,c); d += x; h = x + y;
x = g + f1(d,e,f) + 0xa831c66d + words[25]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0xb00327c8 + words[26]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0xbf597fc7 + words[27]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0xc6e00bf3 + words[28]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0xd5a79147 + words[29]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0x06ca6351 + words[30]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0x14292967 + words[31]; y = f2(b,c,d); e += x; a = x + y;

// extend to 40 words
for (; i < 40; i++)
  words[i] = words[i-16] +
             (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
             words[i-7] +
             (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// fifth round
x = h + f1(e,f,g) + 0x27b70a85 + words[32]; y = f2(a,b,c); d += x; h = x + y;
x = g + f1(d,e,f) + 0x2e1b2138 + words[33]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0x4d2c6dfc + words[34]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0x53380d13 + words[35]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0x650a7354 + words[36]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0x766a0abb + words[37]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0x81c2c92e + words[38]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0x92722c85 + words[39]; y = f2(b,c,d); e += x; a = x + y;

// extend to 48 words
for (; i < 48; i++)
  words[i] = words[i-16] +
             (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
            words[i-7] +
            (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// sixth round
x = h + f1(e,f,g) + 0xa2bfe8a1 + words[40]; y = f2(a,b,c); d += x; h = x + y;
x = g + f1(d,e,f) + 0xa81a664b + words[41]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0xc24b8b70 + words[42]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0xc76c51a3 + words[43]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0xd192e819 + words[44]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0xd6990624 + words[45]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0xf40e3585 + words[46]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0x106aa070 + words[47]; y = f2(b,c,d); e += x; a = x + y;

// extend to 56 words
for (; i < 56; i++)
  words[i] = words[i-16] +
            (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
            words[i-7] +
            (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// seventh round
x = h + f1(e,f,g) + 0x19a4c116 + words[48]; y = f2(a,b,c); d += x; h = x + y;
x = g + f1(d,e,f) + 0x1e376c08 + words[49]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0x2748774c + words[50]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0x34b0bcb5 + words[51]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0x391c0cb3 + words[52]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0x4ed8aa4a + words[53]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0x5b9cca4f + words[54]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0x682e6ff3 + words[55]; y = f2(b,c,d); e += x; a = x + y;

// extend to 64 words
for (; i < 64; i++)
  words[i] = words[i-16] +
            (rotate(words[i-15],  7) ^ rotate(words[i-15], 18) ^ (words[i-15] >>  3)) +
            words[i-7] +
            (rotate(words[i- 2], 17) ^ rotate(words[i- 2], 19) ^ (words[i- 2] >> 10));

// eigth round
x = h + f1(e,f,g) + 0x748f82ee + words[56]; y = f2(a,b,c); d += x; h = x + y;
x = g + f1(d,e,f) + 0x78a5636f + words[57]; y = f2(h,a,b); c += x; g = x + y;
x = f + f1(c,d,e) + 0x84c87814 + words[58]; y = f2(g,h,a); b += x; f = x + y;
x = e + f1(b,c,d) + 0x8cc70208 + words[59]; y = f2(f,g,h); a += x; e = x + y;
x = d + f1(a,b,c) + 0x90befffa + words[60]; y = f2(e,f,g); h += x; d = x + y;
x = c + f1(h,a,b) + 0xa4506ceb + words[61]; y = f2(d,e,f); g += x; c = x + y;
x = b + f1(g,h,a) + 0xbef9a3f7 + words[62]; y = f2(c,d,e); f += x; b = x + y;
x = a + f1(f,g,h) + 0xc67178f2 + words[63]; y = f2(b,c,d); e += x; a = x + y;
```

```
  // update hash
  m_hash[0] += a;
  m_hash[1] += b;
  m_hash[2] += c;
  m_hash[3] += d;
  m_hash[4] += e;
  m_hash[5] += f;
  m_hash[6] += g;
  m_hash[7] += h;
}


/// add arbitrary number of bytes
void SHA256::add(const void* data, size_t numBytes)
{
  const uint8_t* current = (const uint8_t*) data;

  if (m_bufferSize > 0)
  {
    while (numBytes > 0 && m_bufferSize < BlockSize)
    {
      m_buffer[m_bufferSize++] = *current++;
      numBytes--;
    }
  }

  // full buffer
  if (m_bufferSize == BlockSize)
  {
    processBlock(m_buffer);
    m_numBytes  += BlockSize;
    m_bufferSize = 0;
  }

  // no more data ?
  if (numBytes == 0)
    return;

  // process full blocks
  while (numBytes >= BlockSize)
  {
    processBlock(current);
    current    += BlockSize;
    m_numBytes += BlockSize;
    numBytes   -= BlockSize;
```

```
   }

   // keep remaining bytes in buffer
   while (numBytes > 0)
   {
     m_buffer[m_bufferSize++] = *current++;
     numBytes--;
   }
 }


 /// process final block, less than 64 bytes
 void SHA256::processBuffer()
 {
   // the input bytes are considered as bits strings, where the first bit is the most significant
bit of the byte

   // - append "1" bit to message
   // - append "0" bits until message length in bit mod 512 is 448
   // - append length as 64 bit integer

   // number of bits
   size_t paddedLength = m_bufferSize * 8;

   // plus one bit set to 1 (always appended)
   paddedLength++;

   // number of bits must be (numBits % 512) = 448
   size_t lower11Bits = paddedLength & 511;
   if (lower11Bits <= 448)
     paddedLength +=       448 - lower11Bits;
   else
     paddedLength += 512 + 448 - lower11Bits;
   // convert from bits to bytes
   paddedLength /= 8;

   // only needed if additional data flows over into a second block
   unsigned char extra[BlockSize];

   // append a "1" bit, 128 => binary 10000000
   if (m_bufferSize < BlockSize)
     m_buffer[m_bufferSize] = 128;
   else
     extra[0] = 128;

   size_t i;
```

```
  for (i = m_bufferSize + 1; i < BlockSize; i++)
    m_buffer[i] = 0;
  for (; i < paddedLength; i++)
    extra[i - BlockSize] = 0;

  // add message length in bits as 64 bit number
  uint64_t msgBits = 8 * (m_numBytes + m_bufferSize);
  // find right position
  unsigned char* addLength;
  if (paddedLength < BlockSize)
    addLength = m_buffer + paddedLength;
  else
    addLength = extra + paddedLength - BlockSize;

  // must be big endian
  *addLength++ = (unsigned char)((msgBits >> 56) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >> 48) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >> 40) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >> 32) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >> 24) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >> 16) & 0xFF);
  *addLength++ = (unsigned char)((msgBits >>  8) & 0xFF);
  *addLength   = (unsigned char)( msgBits        & 0xFF);

  // process blocks
  processBlock(m_buffer);
  // flowed over into a second block ?
  if (paddedLength > BlockSize)
    processBlock(extra);
}


/// return latest hash as 64 hex characters
std::string SHA256::getHash()
{
  // compute hash (as raw bytes)
  unsigned char rawHash[HashBytes];
  getHash(rawHash);

  // convert to hex string
  std::string result;
  result.reserve(2 * HashBytes);
  for (int i = 0; i < HashBytes; i++)
  {
    static const char dec2hex[16+1] = "0123456789abcdef";
    result += dec2hex[(rawHash[i] >> 4) & 15];
    result += dec2hex[ rawHash[i]       & 15];
  }

  return result;
}


/// return latest hash as bytes
void SHA256::getHash(unsigned char buffer[SHA256::HashBytes])
{
  // save old hash if buffer is partially filled
  uint32_t oldHash[HashValues];
  for (int i = 0; i < HashValues; i++)
    oldHash[i] = m_hash[i];

  // process remaining bytes
  processBuffer();

  unsigned char* current = buffer;
  for (int i = 0; i < HashValues; i++)
  {
    *current++ = (m_hash[i] >> 24) & 0xFF;
    *current++ = (m_hash[i] >> 16) & 0xFF;
    *current++ = (m_hash[i] >>  8) & 0xFF;
    *current++ =  m_hash[i]        & 0xFF;

    // restore old hash
    m_hash[i] = oldHash[i];
  }
}


/// compute SHA256 of a memory block
std::string SHA256::operator()(const void* data, size_t numBytes)
{
  reset();
  add(data, numBytes);
  return getHash();
}


/// compute SHA256 of a string, excluding final zero
std::string SHA256::operator()(const std::string& text)
{
  reset();
  add(text.c_str(), text.size());
```

```
 return getHash();
}
}
```

Б.27 Текст программы «test_snippets\pow_file_watcher.cpp»

```
#include <chrono>
#include <string>
#include <thread>

#include <config/type/fs_watcher.hpp>
#include <modules/fs_watcher.hpp>

#include <utils/logging.hpp>
#include <fmt/core.h>

int main() {
  auto watcher_cfg = config::filesystem_watcher::Default();
  auto watcher = modules::fs_watcher::FileWatcher(watcher_cfg, [](const modules::fs_watcher::FileDiscription& discr, modules::fs_watcher::FileAction act){
    const static auto act_to_string = [](modules::fs_watcher::FileAction& act) {
      switch(act){
        case modules::fs_watcher::FileAction::kCreated:
          return "Created";
        case modules::fs_watcher::FileAction::kModified:
          return "Modified";
        case modules::fs_watcher::FileAction::kDeleted:
          return "Deleted";
      }
    };
    LOG_INFO() << fmt::format("Update> Path: {}, Size: {}, Action: {}\n", discr.path.string(), discr.size, act_to_string(act));
  });

  watcher.List([](const modules::fs_watcher::FileDiscription& discr){
    LOG_INFO() << fmt::format("INIT> Path: {}, Size: {}\n", discr.path.string(), discr.size);
  });

  std::this_thread::sleep_for(std::chrono::minutes(5));
  return 0;
}
```

Б.28 Текст программы «test_snippets\pow_send_receive.cpp»

```
#include <chrono>
#include <string>
#include <thread>

#include <config/type/udp_receiver.hpp>
#include <config/type/udp_sender.hpp>
#include <networking/udp_receiver.hpp>
#include <networking/udp_sender.hpp>

#include <utils/logging.hpp>

int main() {
  auto receiver_cfg = config::udp_receiver::Default();
  auto sender_cfg = config::udp_sender::Default();

  auto receiver = networking::udp_receiver::UdpReceiver(
      receiver_cfg, [](models::udp_buffer::DataBuffer& data) {
        LOG_INFO() << "RECEIVER_CALLBACK: "<< std::string(
                        std::begin(data.buffer),
                        std::begin(data.buffer) + data.last_datagram_size)
                  << "\n";
      });
  auto sender = networking::udp_sender::UdpSender(sender_cfg);

  receiver.Start();
  sender.Start();

  auto payload = std::string("Some data");
  auto data = std::vector<char>(std::begin(payload), std::end(payload));
  sender.Send(std::make_unique<networking::udp_sender::UdpSender::Package>(
      networking::udp_sender::UdpSender::Package{std::move(data), 0}));

  using namespace std::chrono_literals;
  std::this_thread::sleep_for(2000ms);
  return 0;
}
```

Б.29 Текст программы «test_snippets\pow_serialization.cpp»

```
#include <chrono>
#include <string>

#include <helpers/serialization.hpp>
#include <utils/logging.hpp>

int main() {
  const auto serialized = helpers::serialization::Serialize("SErialize Test");
  const auto deserialized = helpers::serialization::Deserialize(serialized.begin(), helpers::serialization::To<std::string>());
  LOG_INFO() << deserialized;
}
```

Б.30 Текст программы «src\config\type\udp_sender.hpp»

```
#pragma once

#include <boost/asio/ip/address.hpp>
#include <boost/asio/ip/udp.hpp>

#include <config/storage.hpp>

namespace config::udp_sender {

namespace ip = boost::asio::ip;

struct Configuration {
  ip::udp ip_version;
  ip::address target_ip;
  ip::port_type target_port;
  int retry_count;
};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}
```

Б.31 Текст программы «src\config\type\udp_receiver.hpp»

```
#pragma once

#include <optional>

#include <boost/asio/ip/address.hpp>
#include <boost/asio/ip/udp.hpp>

#include <config/storage.hpp>

namespace config::udp_receiver {
```

```
namespace ip = boost::asio::ip;

struct Configuration {
  ip::udp ip_version; // udp::v4()
  std::optional<ip::address> target_ip; //ip::address_v4::any()
  ip::port_type target_port;
  int datagram_max_size;
  int buffer_count;
  int receive_threads;
};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}
```

Б.32 Текст программы «src\config\type\fs_watcher.hpp»

```
#pragma once


#include <chrono>
#include <string>
#include <config/storage.hpp>

namespace config::filesystem_watcher {

struct Configuration {
  std::chrono::seconds check_folder_each;
  std::string folder_path;
  uint64_t max_fragment_size;
};

Configuration Get(const ::config::storage::Storage& storage);

Configuration Default();

}
```

Б.33 Текст файла конфигурации «CMakeLists.txt»

```
cmake_minimum_required(VERSION 3.14)
project(one-way-sync VERSION 0.1.0)
```

```
set(CMAKE_CXX_STANDARD 20)

include(FetchContent)

#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})

#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)

set (source_dir "${PROJECT_SOURCE_DIR}/src/")

file (GLOB source_files
    "${source_dir}/*.cpp"
    "${source_dir}/config/type/*.cpp"
    "${source_dir}/config/*.cpp"
    "${source_dir}/helpers/*.cpp"
    "${source_dir}/models/*.cpp"
    "${source_dir}/modules/*.cpp"
    "${source_dir}/networking/*.cpp"
    "${source_dir}/utils/*.cpp"
)
```

```
include_directories(
    "${source_dir}/"
)

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 fmt::fmt-header-only)
target_link_libraries(one-way-sync nlohmann_json::nlohmann_json)
```