Being Restful...

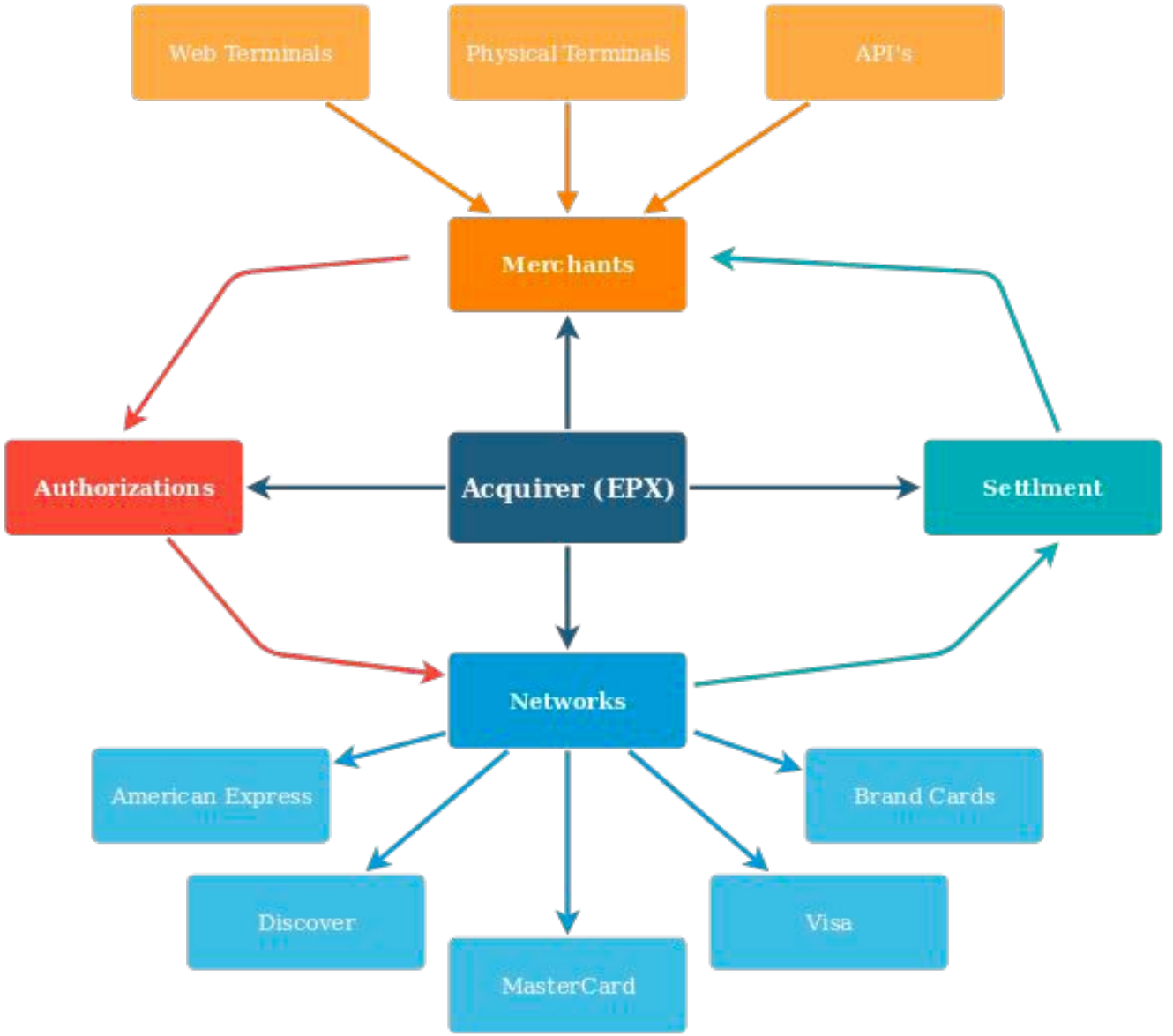
with billions of dollars in transactions, thanks to C++, JSON and HTTP

Background

- Over 30 years in technology
  - Systems/Network Engineer…
  - Online Gaming
  - Financial Modeling
  - Credit Card Processing
- Lead Project Engineer
- Case Study

Electronic Transactions

- Merchant – Coffee Shop, Online Store, Hotel, Government Offices
- Other Processors…
- Acquirer
- Authorization/Settlement
- Various networks…



Overview

- Some History
- What is a RESTful API
  - Basics of the architecture.
  - Simple REST call with result.
- JSON
- HTTP
- C++

HTTP Verbs vs CRUD Operations Quiz

GET

POST

PUT

DELETE

DELETE

UPDATE

READ

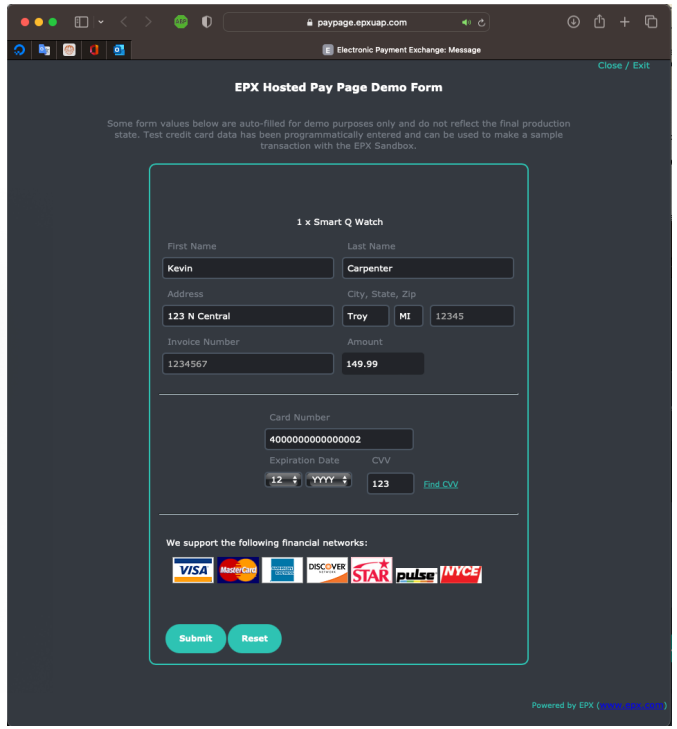
CREATE

What is REST

- Representational State Transfer
- Courtesy of Roy Fielding
- Only adds high level rules.
    - How you implement on the lower level is up to you.
- Six Constraints
    - Client-Server
    - Uniform Interface
    - Stateless
    - Cacheable
    - Layered System
    - Code on Demand

https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Constraints Clients…



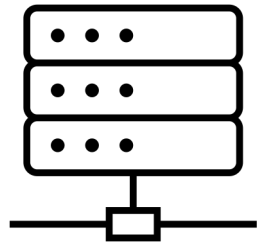
Operation	Verb	URI	Payload	Result
Create	POST	/sale	JSON	Create Record
Read	GET	/batch/<batchID>	Empty	Returns JSON
Update	PUT	/void/<transactionID>	JSON	Updates Record
Delete	DELETE	/sale/<transactionID>	Empty	405 Not Allowed

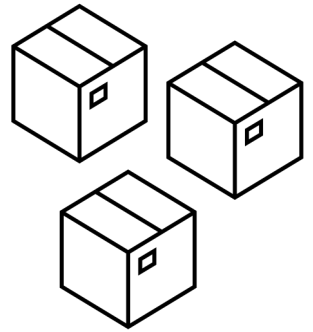
Constraints Server…

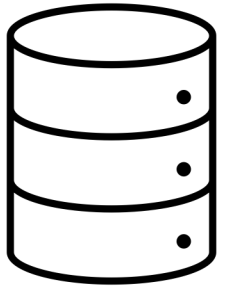
- Uniform Interface
- Stateless
- Cacheable
- Layered System
- Code on Demand

GET /sale/{guid}
POST /sale
PUT /sale/{guid}/capture

GET /refund/{guid}
POST /refund
PUT /void/{guid}







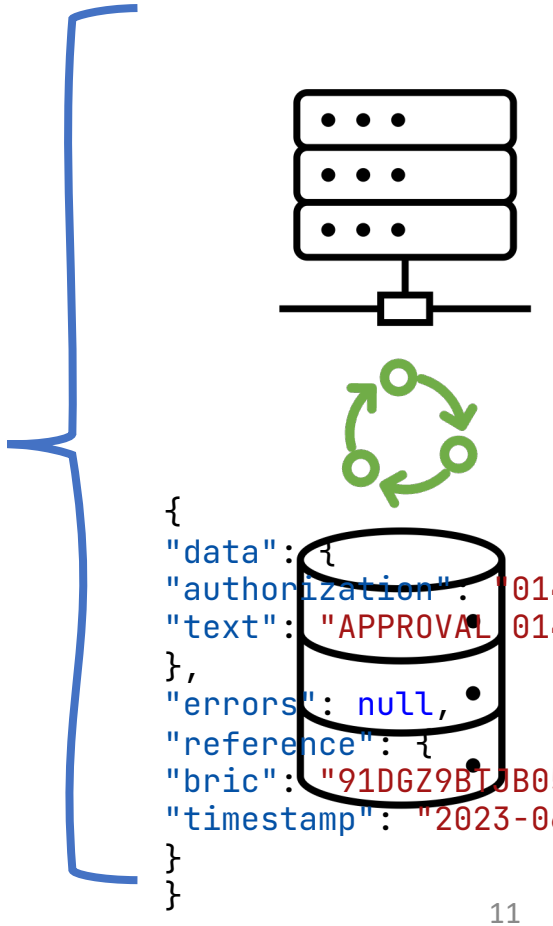
REST Examples

Example POST Transaction

POST http://example.paymentshub.com/sale

```
{
"account": "4111111111111111",
”expiration": ”0524",
"amount": 127.99,
"batchID": 444,
"transaction": 123
}
```

200 OK

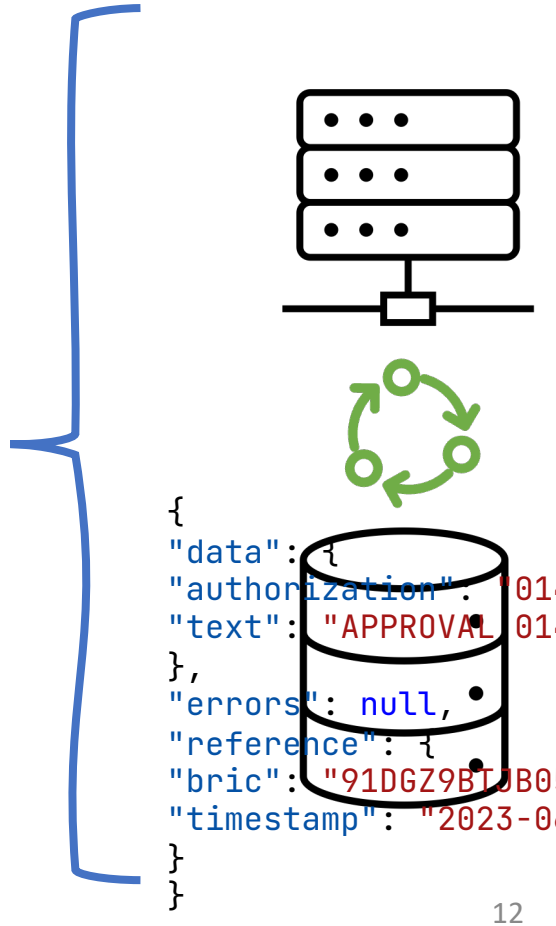


Example PUT Transaction

PUT http://example.paymentshub.com/sale/inc/91DGZ9BTJB05MDGUW02

```
{
"amount": 153.58,
"tipAmount": 25.59
}
```

200 OK

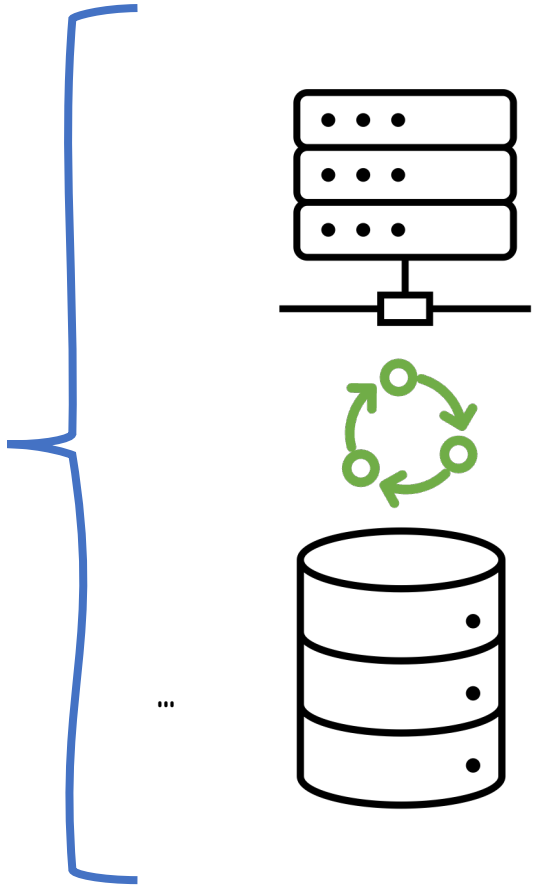


Example DELETE Transaction

DELETE http://example.paymentshub.com/sale/91DGZ9BTJB05MDGUW15

…

405 METHOD NOT ALLOWED



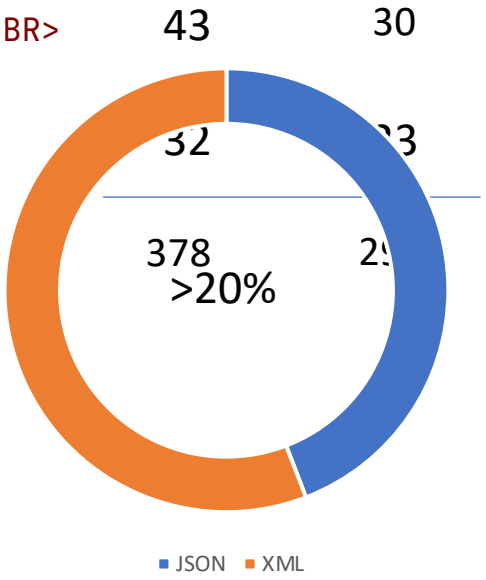
…

Background

- Upgrading existing XML and legacy systems, adding modern JSON/REST API
- Previously using 0MQ – did we change? why?
- Header only please! Why it matters in our environment.
- Performance considerations
- Nlohmann::json – pros and cons
- Cpp-httplib - pros and cons

Why JSON over XML?

```xml
<DETAIL>
<AMOUNT>179.00</AMOUNT>
<BATCH_ID>20230607</BATCH_ID>
<TRAN_NBR>1</TRAN_NBR>
<ACCOUNT_NBR>4111111111111111</ACCOUNT_NBR>
<EXP_DATE>2512</EXP_DATE>
<CARD_ENT_METH>X</CARD_ENT_METH>
<CVV2>123</CVV2>
<FIRST_NAME>Terry</FIRST_NAME>
<LAST_NAME>Tester</LAST_NAME>
<ADDRESS>123 Main Street</ADDRESS>
<CITY>Wilmington</CITY>
<STATE>DE</STATE>
<ZIP_CODE>12345</ZIP_CODE>
</DETAIL>
```




```json
{
"amount": 179.00,
"batchID": 20230607,
"transaction": 1,
"account": "4111111111111111",
"expirationDate": ”2512",
"cardEntryMethod": ”X",
"cvv2": "123",
"address": {
"firstName": ”Terry",
"lastName": ”Tester",
"address": "123 Main Street",
"city": "Wilmington",
"state": "DE",
"zipCode": "12345"
}
}
```

2018 0MQ & JSON

- https://youtu.be/XLSckGMyzbs
- https://github.com/kevinbcarpenter/jz18sub
- 0MQ
- JSON & REST = Peanut Butter & Jelly

Why Header Only?

Size matters.

We are a small team.

Easy to follow.

Minimum dependencies.

Can we maintain it ourselves?

https://github.com/nlohmann/json/releases/tag/v2.0.10

13,000 lines
Now
24,000 lines

https://github.com/yhirose/cpp-httplib/releases/tag/v0.2.0

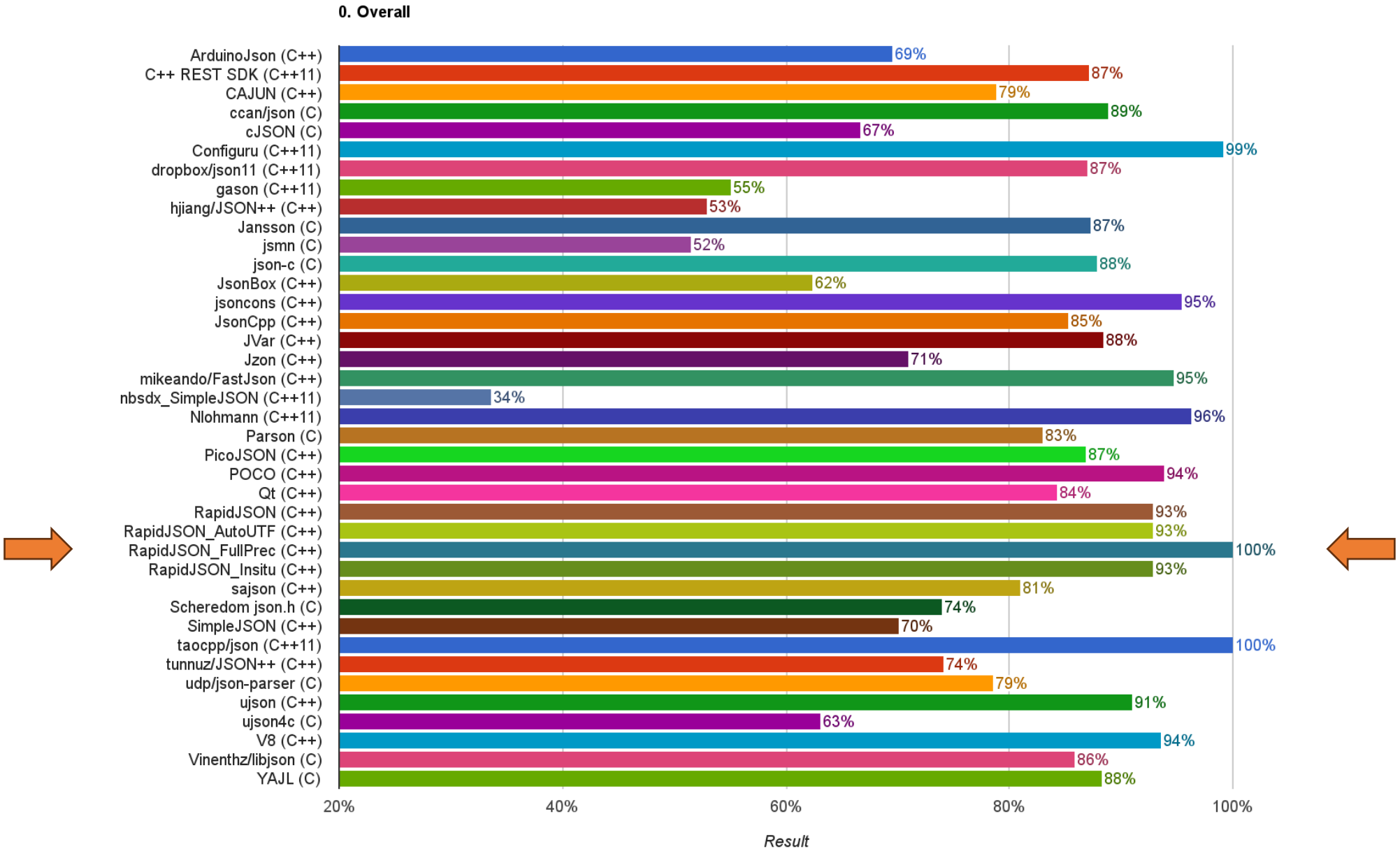
3,000 lines
Now
9,000 lines

JSON for Modern C++

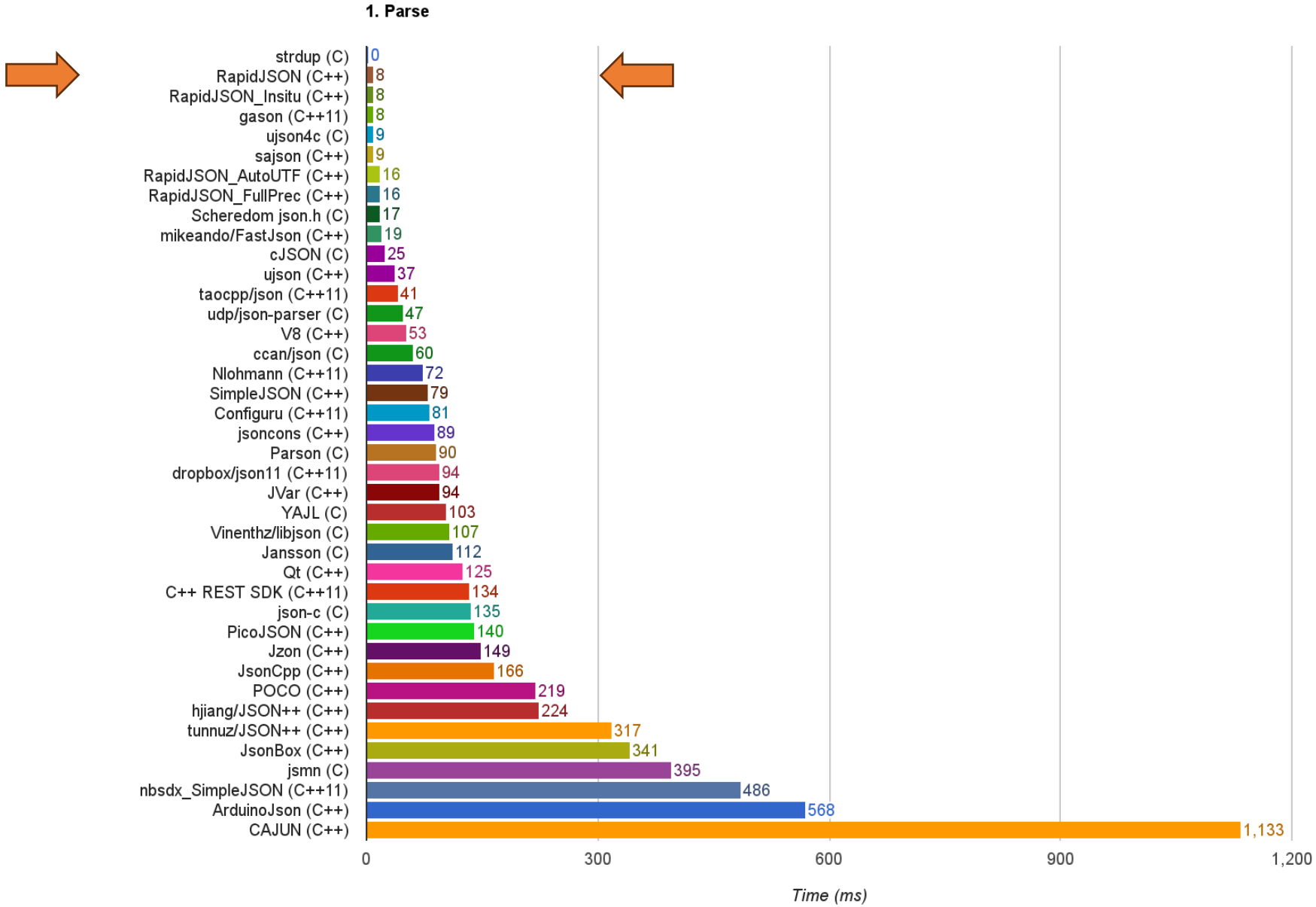
- JavaScript Object Notation
- Massive language support
- Thanks Niels Lohmann
- github.com/nlohmann/json

How to compare JSON libraries?
https://github.com/miloyip/nativejson-benchmark

No comments, No Error Handling, No Date Type, not as Robust as XML



Conformance



Parsing Time

cpp-httplib

- HTTP Library
- Client / Server
- github.com/yhirose/cpp-httplib

Blocking vs non-blocking

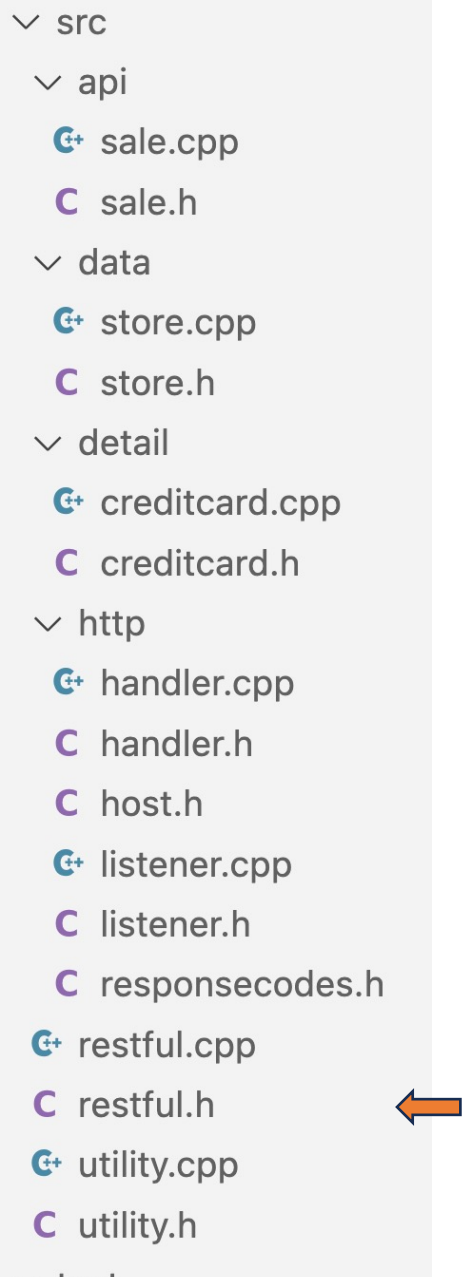
https://github.com/guteksan/REST-CPP-benchmark

```
#define CPPHTTPLIB_OPENSSL_SUPPORT
```

The Approach

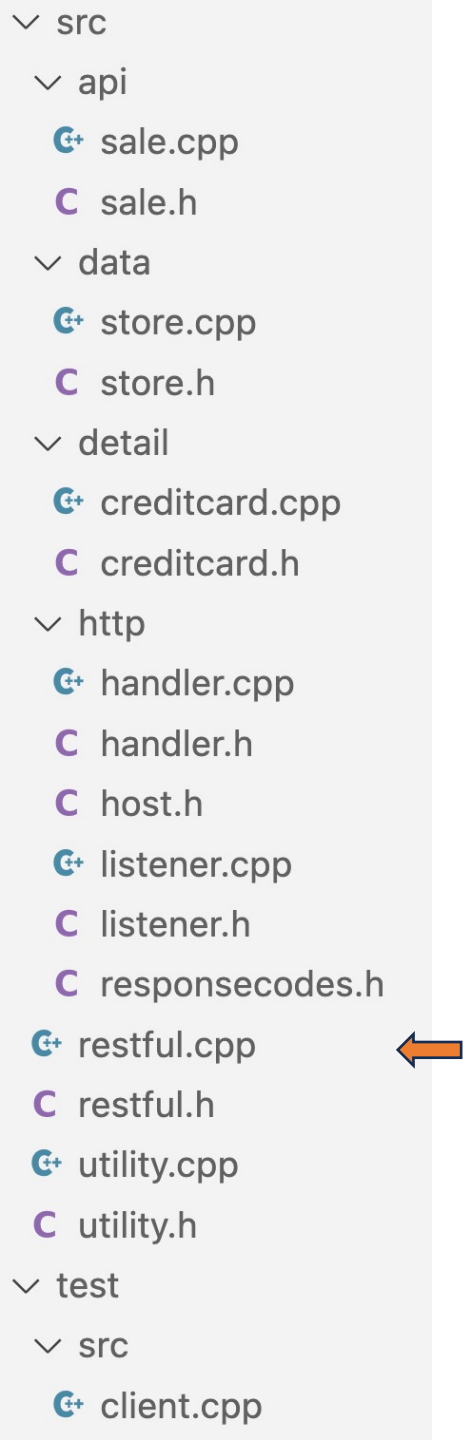
- Basic HTTP Server
- REST Practices!
- Creating HTTP Client
- Lessons Learned

On to the code!

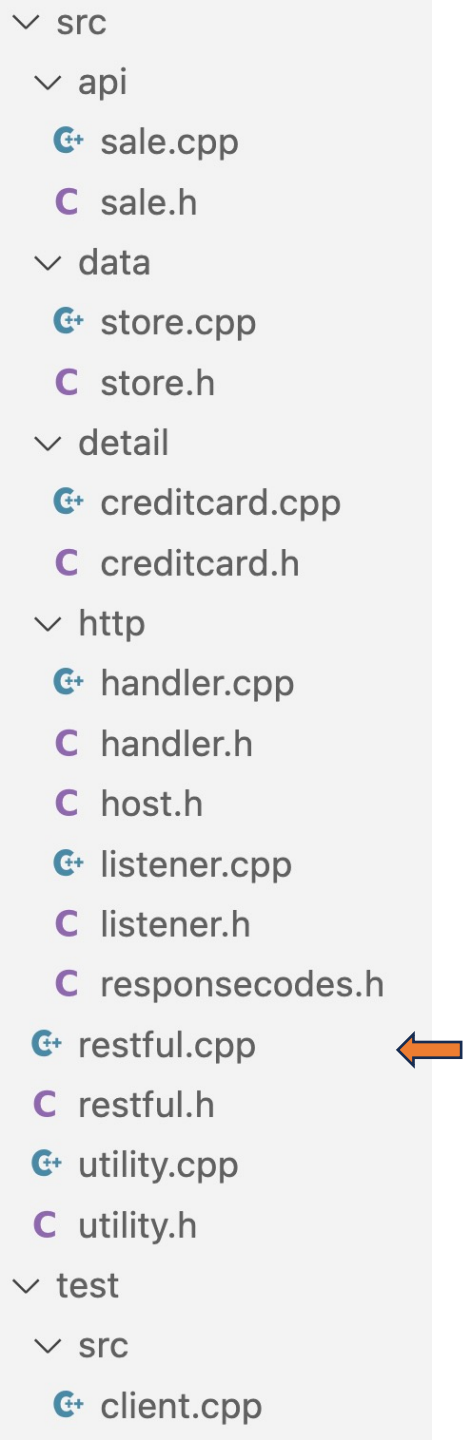



```cpp
#pragma once
#include <App.h>

#include <json.hpp>

class Restful : public App {
private:
  void setup(const json &configuration) override;
  void start();
  void idle();

public:
  Restful(std::string appName);
  ~Restful() final = default;
  void run() override;
};
```

Scaffolding / Supporting Class



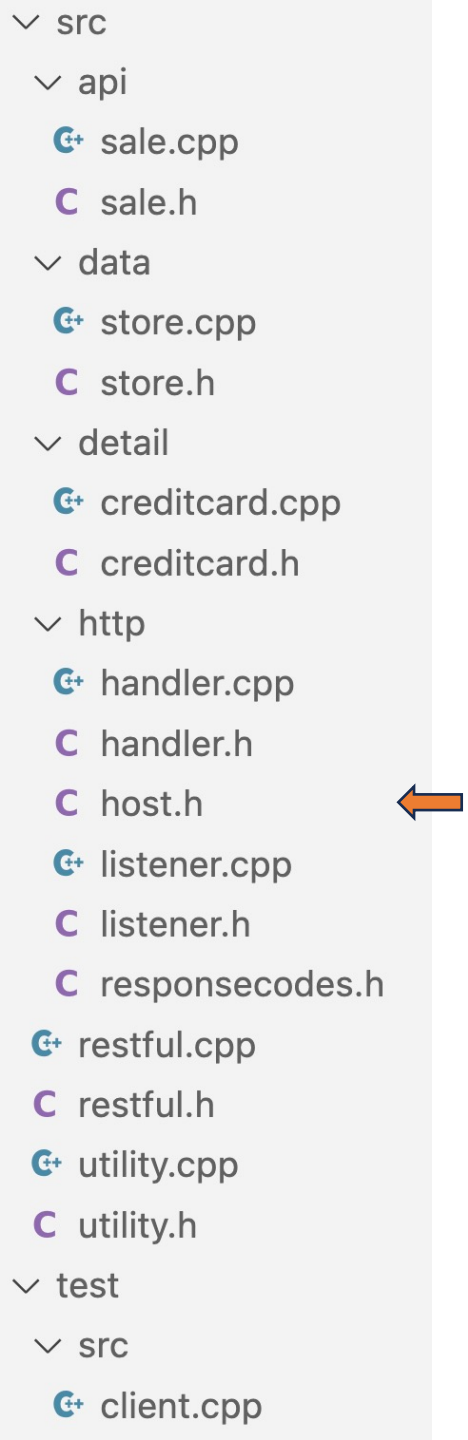

```cpp
namespace {
const std::string VERSION{"0.3"};
const std::string APP_NAME{"restful"};
auto logline(const std::string &func) { return "Restful: " + func; }
} // namespace

//------------------------------------------------------------------------
void Restful::idle() {
  const auto fname = ::logline(static_cast<const char *>(__func__));
  constexpr int idleTime = 5;
  while (true) {
    std::this_thread::sleep_for(std::chrono::seconds(idleTime));
    Logger::getLogger()->save(fname, "idle...", IS_MAIN);
  }
}

//------------------------------------------------------------------------
void Restful::run() {
  App::run();
  start();
  idle();
}

//------------------------------------------------------------------------
auto main(int argc, char *argv[]) -> int {
  App::main(argc, argv, VERSION, new Restful(APP_NAME));
  return 0;
}
```




```cpp
//------------------------------------------------------------------------
void Restful::start() {
  const auto fname = ::logline(static_cast<const char *>(__func__));

  http::host host;
  host.setup(App::getConfig().at("httpServer"));

  std::ostringstream os;
  os << "Starting server v" << VERSION << " on " << host.ip << ":" << host.port
     << std::endl;

  std::thread(&http::listener::run, std::move(host)).detach();

  Logger::getLogger()->save(fname, os.str(), IS_MAIN);
}
```

restful.json

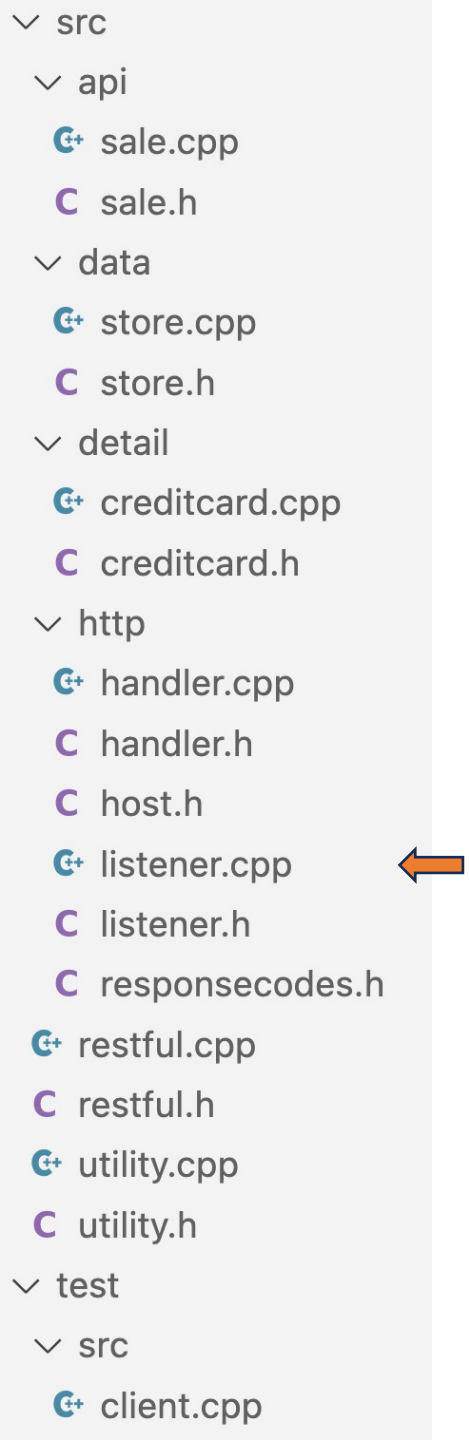
```
"httpServer": {
  "ip": "0.0.0.0",
  "port": 8080,
  "bannedIps": [
    "192.168.7.2",
    "192.178.7.4"
  ]
}
```




```cpp
#include <json.hpp>
#include <string>
#include <unordered_set>

using json = nlohmann::json;

namespace http {
struct host {
  std::string ip{};
  uint port;
  std::unordered_set<std::string> bannedips;

  void setup(const json &config) {
    ip = config.at("ip");
    port = config.at("port");
    if (config.contains("bannedIps")) {
      bannedips = config.value("bannedIps",std::unordered_set<std::string>{});
    }
  }

  bool isBanned(const std::string &ipaddr) const {
    return bannedips.find(ipaddr) != bannedips.end();
  }
};

} // namespace http
```

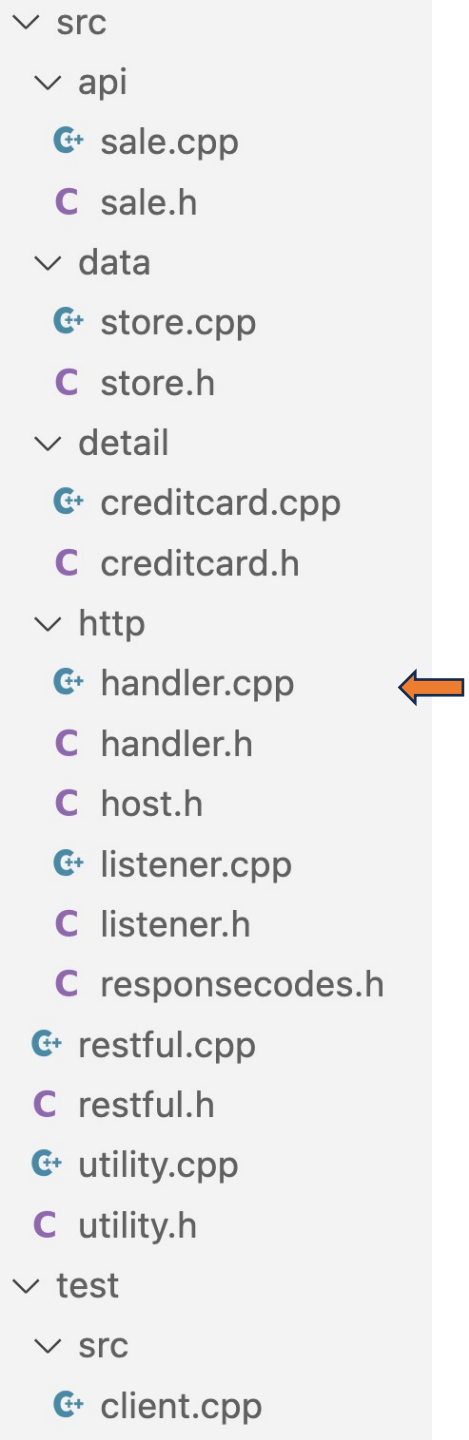
restful.json

```
"httpServer": {
  "ip": "0.0.0.0",
  "port": 8080,
  "bannedIps": [
    "192.168.7.2",
    "192.178.7.4"
  ]
}
```

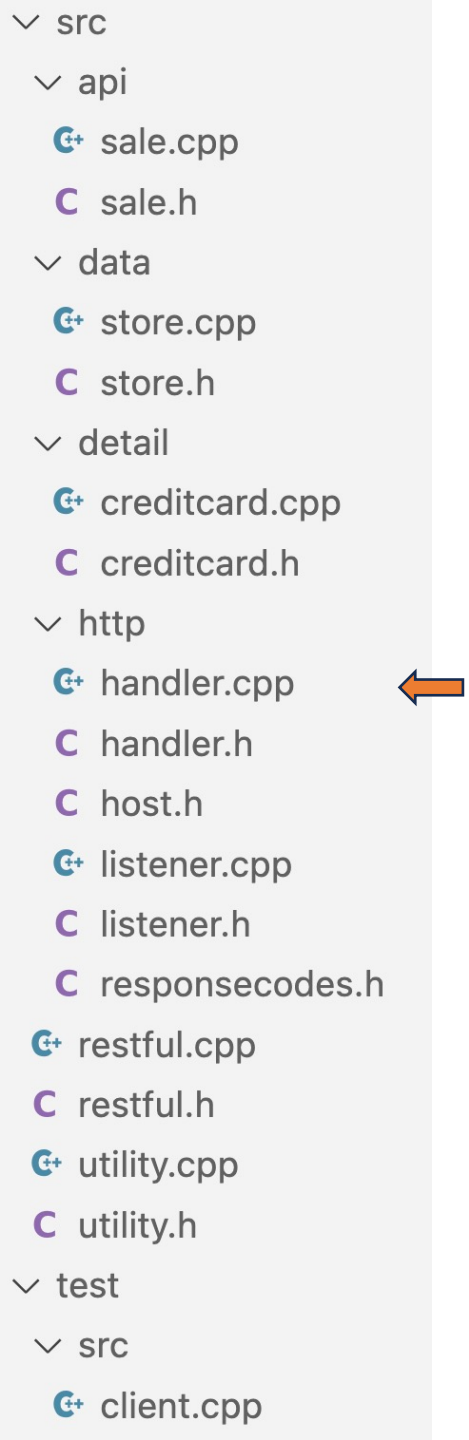



```cpp
#include "listener.h"
#include "handler.h"
#include "httplib.h"

namespace http::listener {

void run(host host) {
  httplib::Server srv;
  handler::setup(srv, host);
  srv.listen(host.ip, host.port);
}

} // namespace http::listener
```

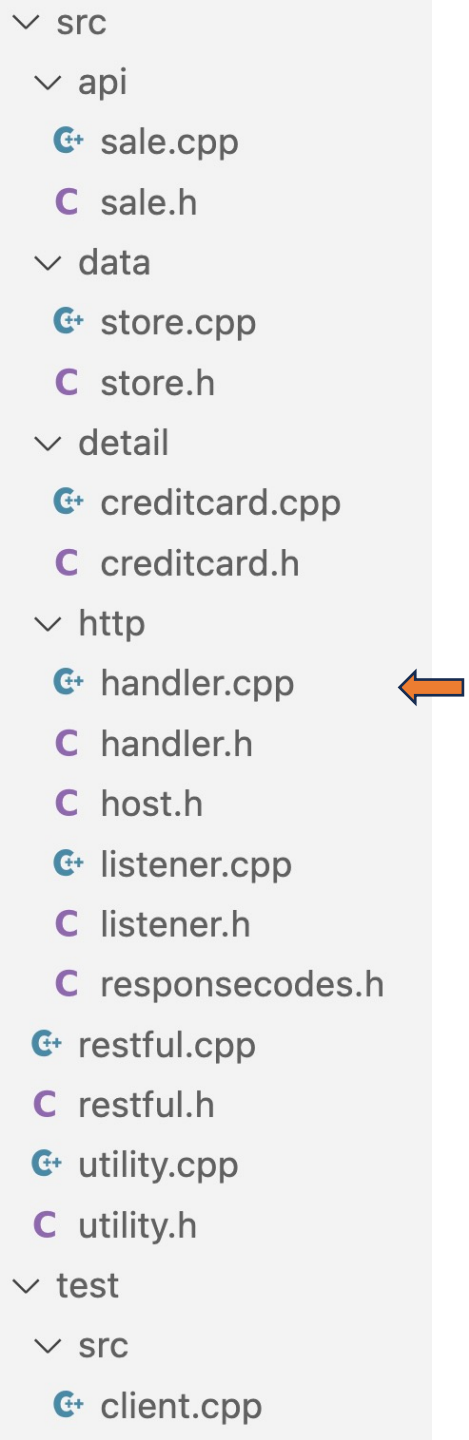
```cpp
std::thread(&http::listener::run, std::move(host)).detach();
```




```cpp
namespace http::handler {
auto logline(const std::string &function) { return "handler::" + function; }

//-----------------------------------------------------------------------
void security(httplib::Server &server, const host &host) {…}

//-----------------------------------------------------------------------
void errors(httplib::Server &server) {…}

//-----------------------------------------------------------------------
void routing(httplib::Server &server) {…}


//-----------------------------------------------------------------------
void setup(httplib::Server &srv, const host &host) {
  security(srv, host);
  errors(srv);
  routing(srv);
}

} // namespace http::handler
```

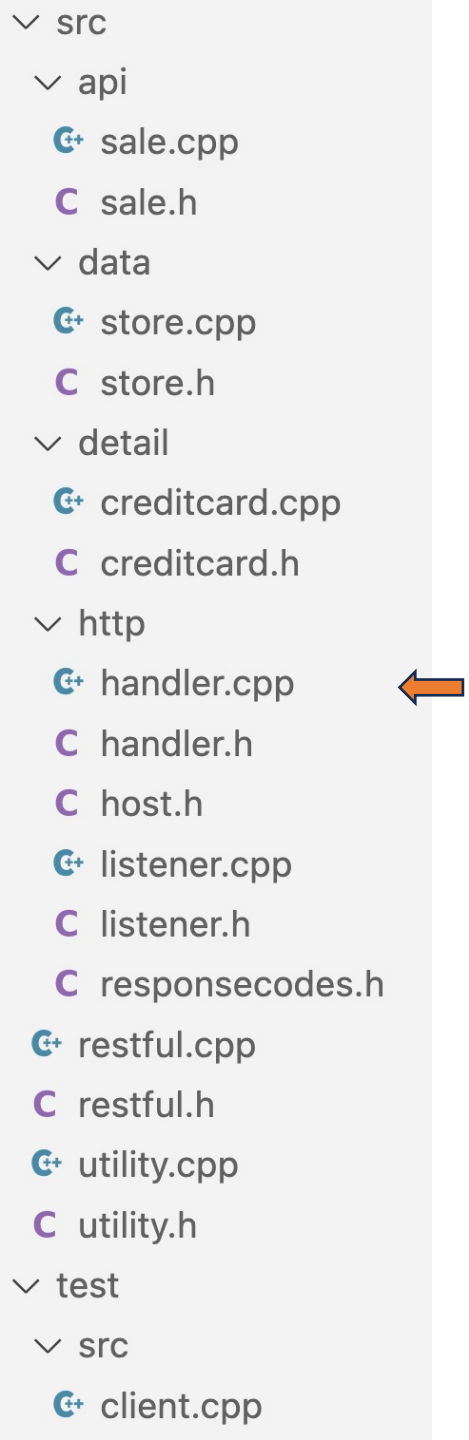



```cpp
//------------------------------------------------------------------------------
void security(httplib::Server &server, const host &host) {…}

//------------------------------------------------------------------------------
void errors(httplib::Server &server) {…}

//------------------------------------------------------------------------------
void routing(httplib::Server &server) {
  server.Post("/sale",
              [](const httplib::Request &req, httplib::Response &res) {
                 api::sale::auth(req, res);
  });

  server.Put(R"(^/sale/inc/(\w+)$)",
              [](const httplib::Request &req, httplib::Response &res) {
                 api::sale::inc(req, res);
  });
  server.Get("/sale",
              [](const httplib::Request &req, httplib::Response &res) -> void {
                 api::sale::list(req, res);
  });
  server.Delete(R"(^/sale/void/(\w+)$)",
              [](const httplib::Request &req, httplib::Response &res) {
              api::sale::reverse(req, res);
  });
}

} // namespace http::handler
```

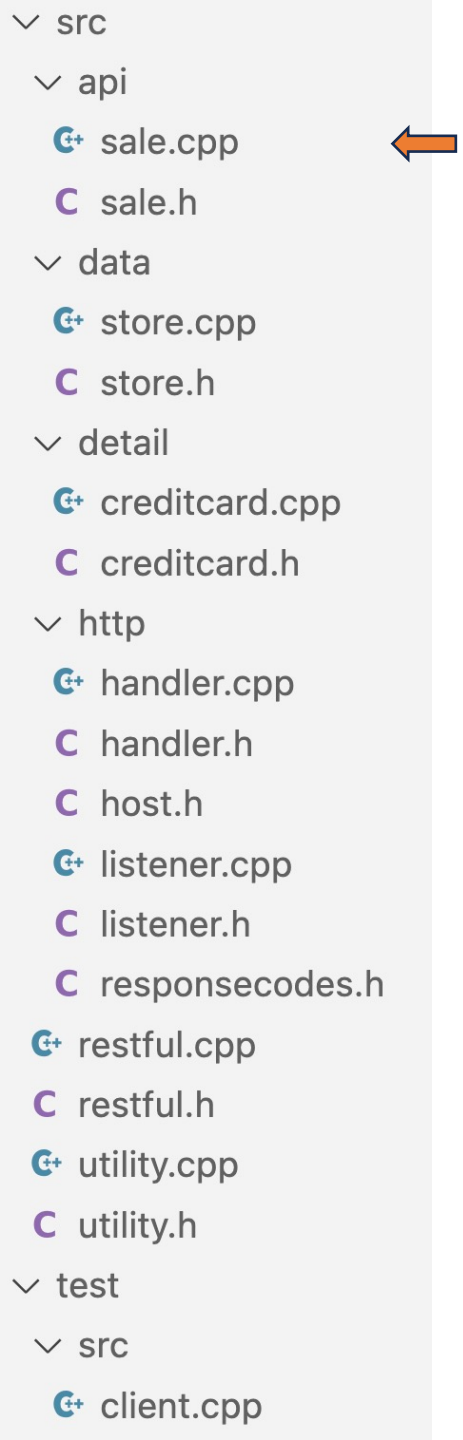



```cpp
//------------------------------------------------------------------------
void security(httplib::Server &server, const host &host) {
  const auto fName = logline(static_cast<const char *>(__func__));
  static std::atomic<uint> cid(0);

  server.set_pre_routing_handler([&host, fName](auto &req, auto &res) {
    req.setId(++cid);
    auto ipaddr = req.get_header_value("REMOTE_ADDR");
    auto useragent = req.get_header_value("User-Agent");

    if (host.isBanned(ipaddr)) {
      res.status = code::Unauthorized;
      Logger::getLogger()->warn(fName, "IP Blocked: " + ipaddr, IS_THREAD,
      req.id());
      return httplib::Server::HandlerResponse::Handled;
    }

    return httplib::Server::HandlerResponse::Unhandled;
  });
}

//------------------------------------------------------------------------
void errors(httplib::Server &server) {…}

//------------------------------------------------------------------------
void routing(httplib::Server &server) {…}

} // namespace http::handler
```

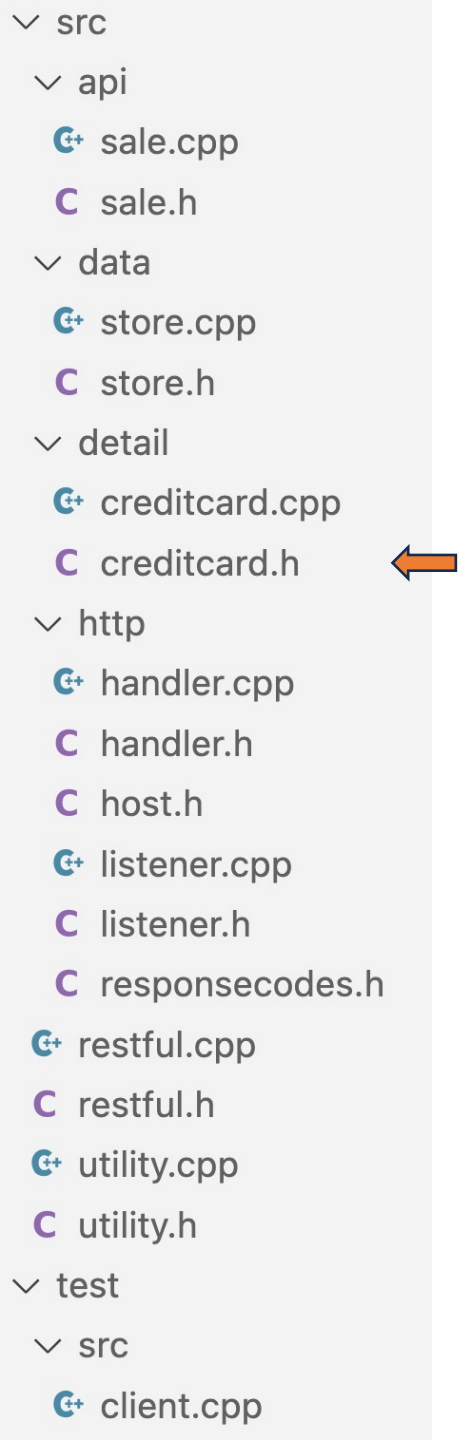
set_post_routing_handler



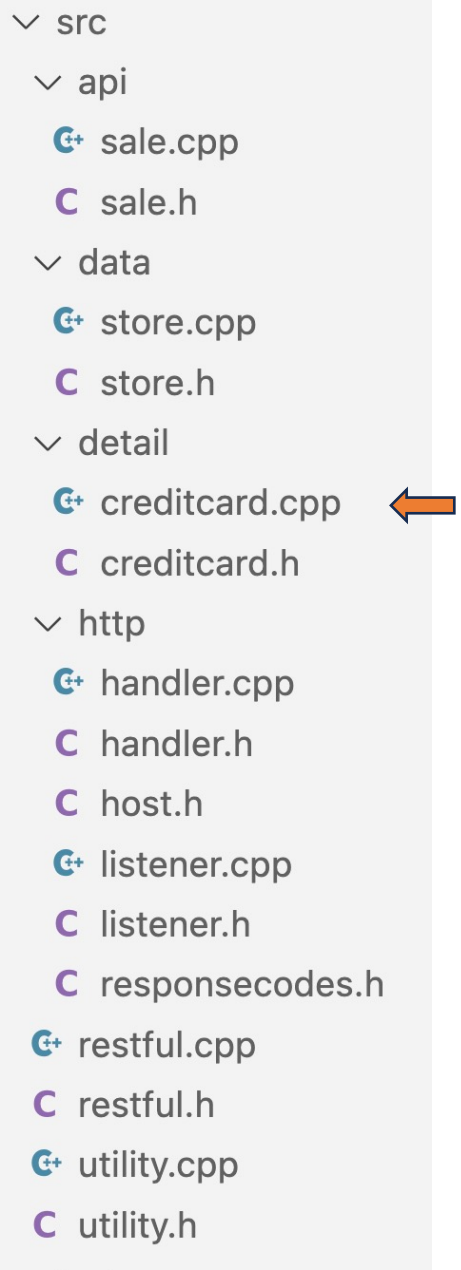

```cpp
//-----------------------------------------------------------------------
void security(httplib::Server &server, const host &host) {…}

//-----------------------------------------------------------------------
void errors(httplib::Server &server) {
  const auto fName = logline(static_cast<const char *>(__func__));

  server.set_error_handler([fName](const auto &req, auto &res) {
    auto ipAddr = "IP:" + req.get_header_value("REMOTE_ADDR") + " ";
    auto methodAndPath = req.method + " " + req.path;
    auto status = " " + std::to_string(res.status) + "";
    auto logStr = ipAddr + methodAndPath + status;

    Logger::getLogger()->warn(fName, logStr, IS_THREAD, req.id());
    Logger::getLogger()->warn(fName, req.body, IS_THREAD, req.id());
  });
}

//-----------------------------------------------------------------------
void routing(httplib::Server &server) {…}

} // namespace http::handler
```




```cpp
namespace api::sale {
constexpr auto contentType = "application/json";

auto logline(const std::string &func) -> std::string {
return "api::sale::" + func;
}


//-----------------------------------------------------------------------
void auth(const httplib::Request &req, httplib::Response &res) {
  const auto fName = logline(static_cast<const char *>(__func__));
  Logger *log = Logger::getLogger();

  detail::cardInfo ci = json::parse(req.body);
  ci.guid = utility::guid();
  data::store::save(ci.guid, ci);

  log->save(fName, "Posted Sale", IS_THREAD, req.id());

  res.status = http::code::OK;

  json result{ci};
  res.set_content(result.dump(), contentType);
};
```

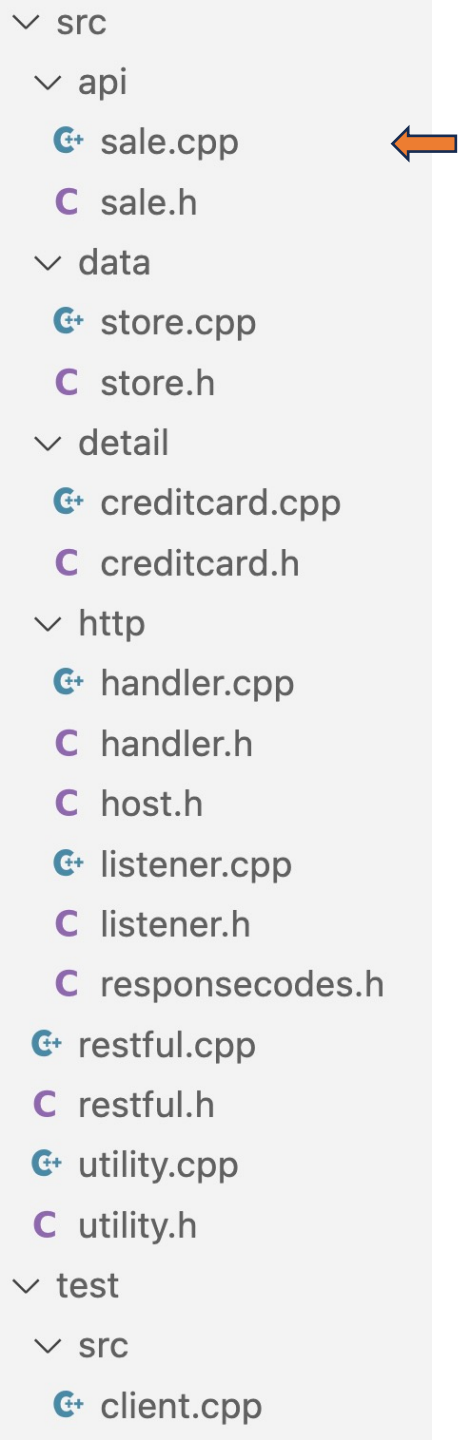



```cpp
#pragma once

#include "json.hpp"
#include <string>

using json = nlohmann::json;

namespace detail {

struct cardInfo {
  std::string account{};
  std::string expiration{};
  double amount{0.0};
  double tipAmount{0.0};
  uint batchId{0};
  uint transaction{0};
  std::string guid{};
};

void from_json(const json &j, cardInfo &c);
void to_json(json &j, const cardInfo &c);

} // namespace detail
```



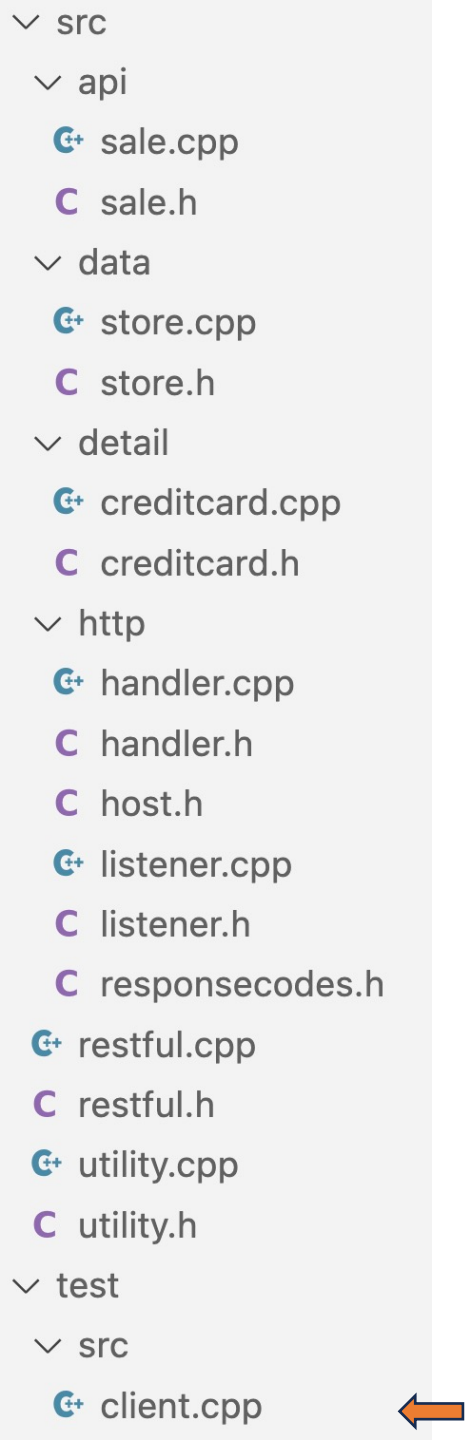

```cpp
#include "creditcard.h"

namespace detail {

void from_json(const json &j, cardInfo &c) {
  c.account = j.value("account", "");
  c.expiration = j.value("expiration", "");
  c.guid = j.value("guid", "");
  c.amount = j.value("amount", 0.0);
  c.tipAmount = j.value("tipAmount", 0.0);
  c.batchId = j.value("batchId", 0);
  c.transaction = j.value("transaction", 0);
}

void to_json(json &j, const cardInfo &c) {
  j = json{{"account", c.account},
           {"expiration", c.expiration},
           {"guid", c.guid},
           {"amount", c.amount},
           {"tipAmount", c.tipAmount},
           {"batchId", c.batchId},
           {"transaction", c.transaction}};
};

} // namespace detail
```

NLOHMANN_DEFINE_TYPE_INTRUSIVE(account, expiration…)

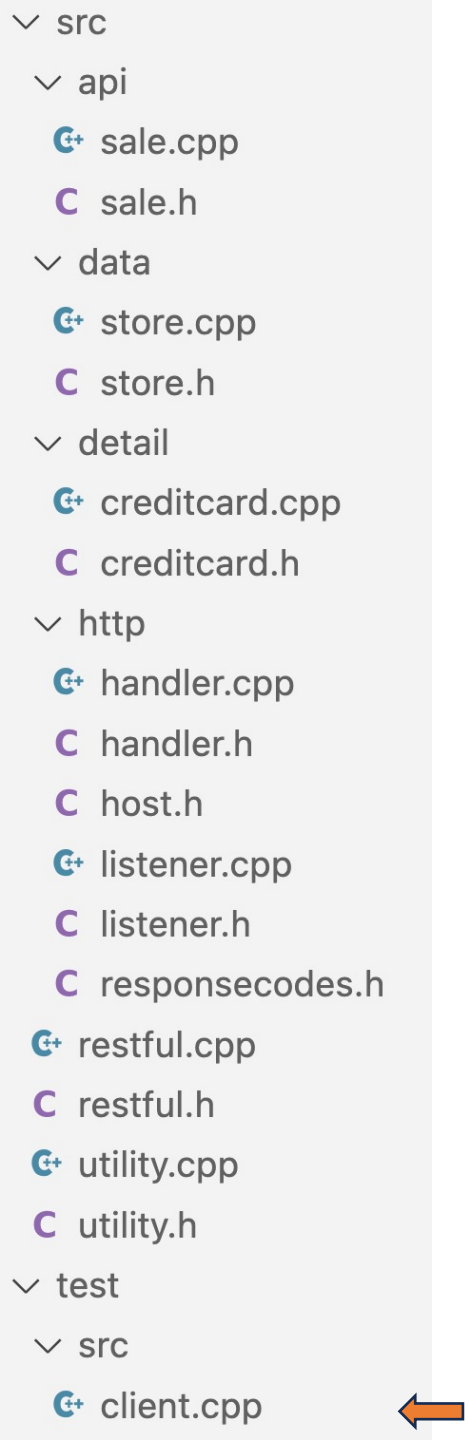


server.Put(R"(^/sale/inc/(\w+)$)"…

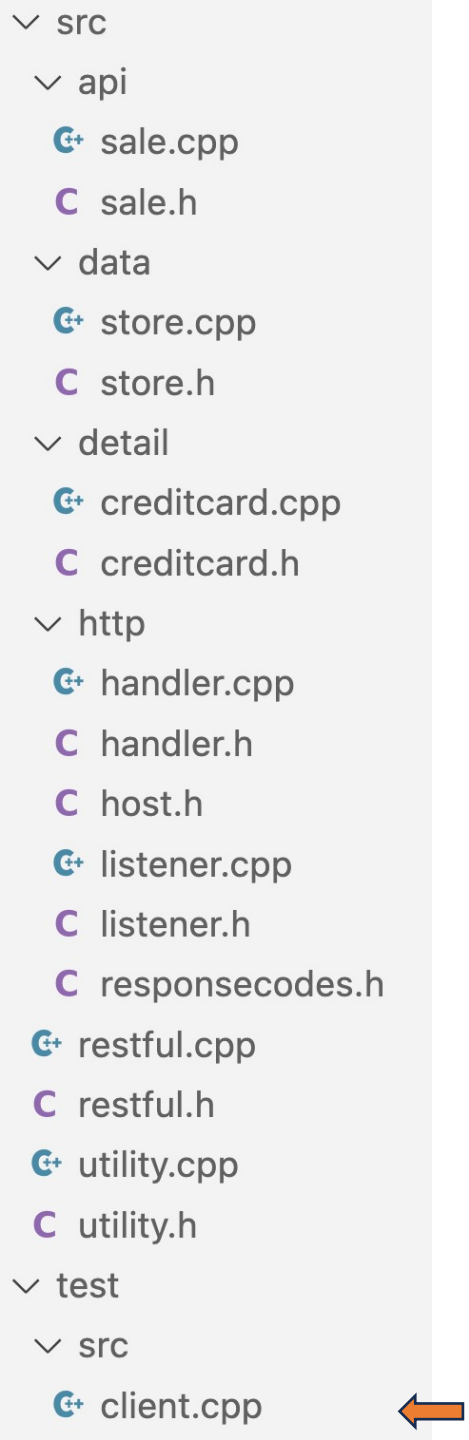
```
//----------------------------------------------------------------------
void inc(const httplib::Request &req, httplib::Response &res) {
  const auto fName = logline(static_cast<const char *>(__func__));
  Logger *log = Logger::getLogger();
  json result{};

  detail::cardInfo uci = json::parse(req.body);
  auto guid = req.matches[1];
  auto sci = data::store::get(guid);

  if (sci.has_value()) {
    if (uci.amount != (uci.tipAmount + sci->amount)) {
      log->warn(fName, "New amount with tip does not equal differs.", IS_THREAD,
      req.id());
    }
    sci->tipAmount = uci.tipAmount;
    sci->amount = uci.amount;

    log->save(fName, "Incremented Sale", IS_THREAD, req.id());
    res.status = http::code::OK;
    json result = sci.value();
  } else {
    log->warn(fName, "Transaction not found.", IS_THREAD, req.id());
    res.status = http::code::InternalServerError;
  }

  res.set_content(result.dump(), contentType);
};
```




```cpp
#include "detail/creditcard.h"
#include <json.hpp>
using json = nlohmann::json;

constexpr auto TESTCOUNT = 5;
constexpr auto DELAY = 1;
constexpr auto contentType = "application/json";

//----------------------------------------------------------------------
auto main(int /* argc*/, char ** /*argv[]*/) -> int {
  uint idleTime = DELAY;
  httplib::Client cli("http://127.0.0.1:8080");

  for (int i = 0; i < TESTCOUNT; ++i) {
    detail::cardInfo c1{"4111111111111111", "0524", 125.00, 0, 1, 1, ""};
    auto c2 = testPost(cli, c1);
    testPut(cli, c2);
    std::this_thread::sleep_for(std::chrono::seconds(idleTime));
  }

  auto sales = testGet(cli);
  for (auto s : sales) {
    auto d1 = testRemove(cli, s.guid);
    std::this_thread::sleep_for(std::chrono::seconds(idleTime));
  }

  return 0;
}
```




```cpp
//---------------------------------------------------------------------------
auto testPost(httplib::Client &cli, detail::cardInfo &ci) -> detail::cardInfo {
  json payload = ci;
  auto res = cli.Post("/sale", payload.dump(), contentType);

  detail::cardInfo rci = json::parse(res->body)[0];
  std::cout << "POST/CREATE - Status: " << res->status << "Guid: " << rci.guid
            << std::endl
            << "Body: " << res->body << std::endl;

  return rci;
}

//---------------------------------------------------------------------------
void testPut(httplib::Client &cli, detail::cardInfo &ci) {
  ci.tipAmount = ci.amount * .20;
  ci.amount += ci.tipAmount;
  json payload = ci;

  std::stringstream putPath{};
  putPath << "/sale/inc/" << ci.guid;

  auto res = cli.Put(putPath.str(), payload.dump(), contentType);

  std::cout << "PUT/UPDATE - Status: " << res->status << std::endl;
}
```

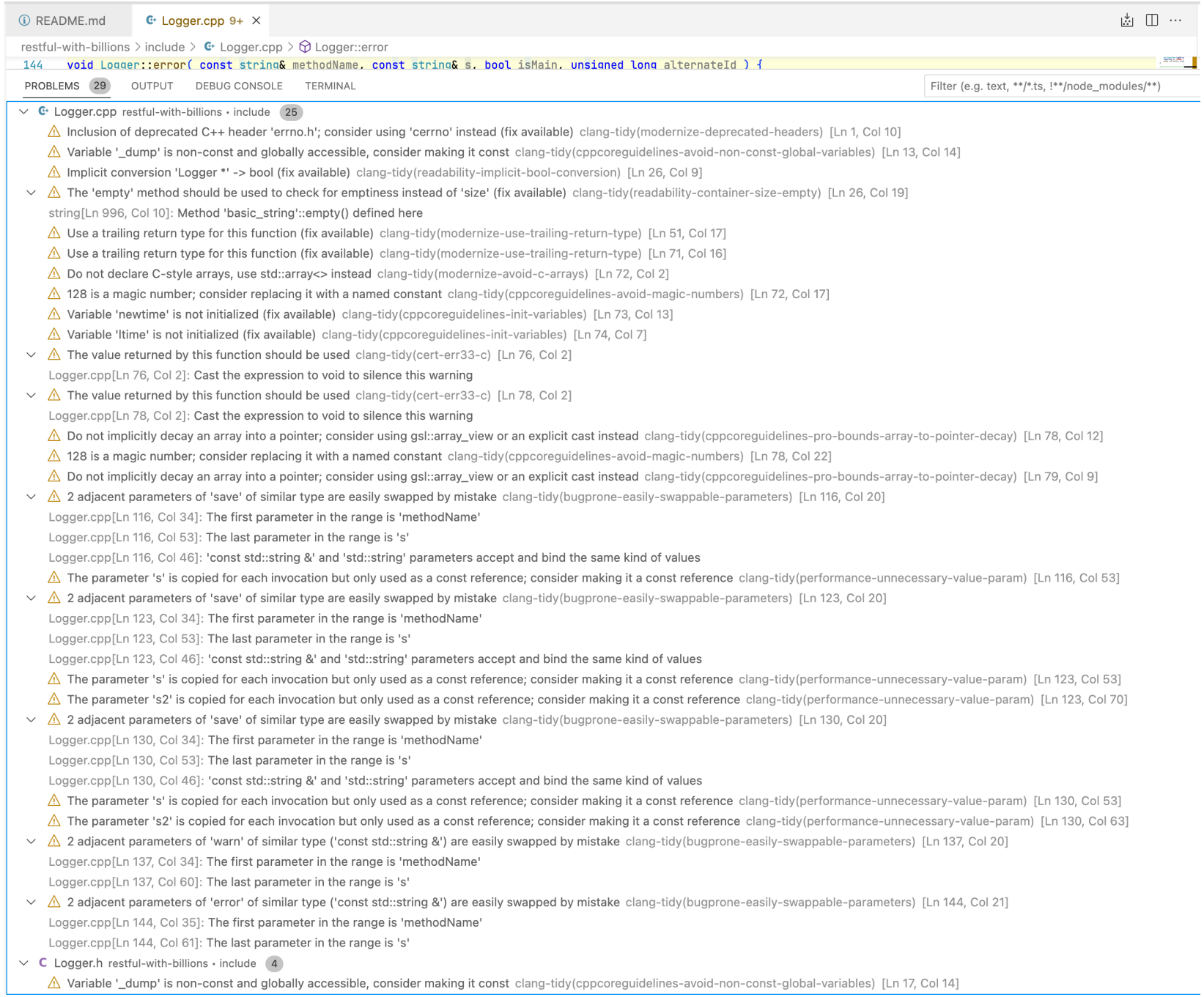



```cpp
//-----------------------------------------------------------------------
auto testGet(httplib::Client &cli) -> std::vector<detail::cardInfo> {
  std::cout << "RESULT from GET" << std::endl;
  auto res = cli.Get("/sale");

  std::cout << "GET/LIST - Status: " << res->status << std::endl
            << "Body: " << res->body << std::endl;

  auto trans = json::parse(res->body);
  return trans;
}

//-----------------------------------------------------------------------
auto testRemove(httplib::Client &cli, std::string guid) {
  std::stringstream delPath{};
  delPath << "/sale/void/" << guid;

  auto res = cli.Delete(delPath.str());
  std::cout << "DELETE/DELETE - Status: " << res->status << std::endl;

  return res;
}
```


Live Demo?

Singleton or Injection

- Our Logger.h class remains a singleton'ish
  - Good for writing output.
  - Easily reachable.
  - Interleaving is not an issue for us.
- Injection solved some non-const global static issues.
  - Configuration objects used by client connections.
  - Could be troublesome if another user changed.
  - Configuration can reload dynamically.
  - Was bad in a network interface binding instance.


README.md
Logger.cpp 9+
restful-with-billions > include > Logger.cpp > Logger::error
144 void Logger::error(const string& methodName, const string& s, bool isMain, unsigned long alternateId) {
PROBLEMS 29
OUTPUT
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, **/*.ts, !**/node_modules/**)
Logger.cpp restful-with-billions • include 25
Inclusion of deprecated C++ header 'errno.h'; consider using 'cerrno' instead (fix available) clang-tidy(modernize-deprecated-headers) [Ln 1, Col 10]
Variable '_dump' is non-const and globally accessible, consider making it const clang-tidy(cppcoreguidelines-avoid-non-const-global-variables) [Ln 13, Col 14]
Implicit conversion 'Logger *' -> bool (fix available) clang-tidy(readability-implicit-bool-conversion) [Ln 26, Col 9]
The 'empty' method should be used to check for emptiness instead of 'size' (fix available) clang-tidy(readability-container-size-empty) [Ln 26, Col 19]
string[Ln 996, Col 10]: Method 'basic_string'::empty() defined here
Use a trailing return type for this function (fix available) clang-tidy(modernize-use-trailing-return-type) [Ln 51, Col 17]
Use a trailing return type for this function (fix available) clang-tidy(modernize-use-trailing-return-type) [Ln 71, Col 16]
Do not declare C-style arrays, use std::array<> instead clang-tidy(modernize-avoid-c-arrays) [Ln 72, Col 2]
128 is a magic number; consider replacing it with a named constant clang-tidy(cppcoreguidelines-avoid-magic-numbers) [Ln 72, Col 17]
Variable 'newtime' is not initialized (fix available) clang-tidy(cppcoreguidelines-init-variables) [Ln 73, Col 13]
Variable 'ltime' is not initialized (fix available) clang-tidy(cppcoreguidelines-init-variables) [Ln 74, Col 7]
The value returned by this function should be used clang-tidy(cert-err33-c) [Ln 76, Col 2]
Logger.cpp[Ln 76, Col 2]: Cast the expression to void to silence this warning
The value returned by this function should be used clang-tidy(cert-err33-c) [Ln 78, Col 2]
Logger.cpp[Ln 78, Col 2]: Cast the expression to void to silence this warning
Do not implicitly decay an array into a pointer; consider using gsl::array_view or an explicit cast instead clang-tidy(cppcoreguidelines-pro-bounds-array-to-pointer-decay) [Ln 78, Col 12]
128 is a magic number; consider replacing it with a named constant clang-tidy(cppcoreguidelines-avoid-magic-numbers) [Ln 78, Col 22]
Do not implicitly decay an array into a pointer; consider using gsl::array_view or an explicit cast instead clang-tidy(cppcoreguidelines-pro-bounds-array-to-pointer-decay) [Ln 79, Col 9]
2 adjacent parameters of 'save' of similar type are easily swapped by mistake clang-tidy(bugprone-easily-swappable-parameters) [Ln 116, Col 20]
Logger.cpp[Ln 116, Col 34]: The first parameter in the range is 'methodName'
Logger.cpp[Ln 116, Col 53]: The last parameter in the range is 's'
Logger.cpp[Ln 116, Col 46]: 'const std::string &' and 'std::string' parameters accept and bind the same kind of values
The parameter 's' is copied for each invocation but only used as a const reference; consider making it a const reference clang-tidy(performance-unnecessary-value-param) [Ln 116, Col 53]
2 adjacent parameters of 'save' of similar type are easily swapped by mistake clang-tidy(bugprone-easily-swappable-parameters) [Ln 123, Col 20]
Logger.cpp[Ln 123, Col 34]: The first parameter in the range is 'methodName'
Logger.cpp[Ln 123, Col 53]: The last parameter in the range is 's'
Logger.cpp[Ln 123, Col 46]: 'const std::string &' and 'std::string' parameters accept and bind the same kind of values
The parameter 's' is copied for each invocation but only used as a const reference; consider making it a const reference clang-tidy(performance-unnecessary-value-param) [Ln 123, Col 53]
The parameter 's2' is copied for each invocation but only used as a const reference; consider making it a const reference clang-tidy(performance-unnecessary-value-param) [Ln 123, Col 70]
2 adjacent parameters of 'save' of similar type are easily swapped by mistake clang-tidy(bugprone-easily-swappable-parameters) [Ln 130, Col 20]
Logger.cpp[Ln 130, Col 34]: The first parameter in the range is 'methodName'
Logger.cpp[Ln 130, Col 53]: The last parameter in the range is 's'
Logger.cpp[Ln 130, Col 46]: 'const std::string &' and 'std::string' parameters accept and bind the same kind of values
The parameter 's' is copied for each invocation but only used as a const reference; consider making it a const reference clang-tidy(performance-unnecessary-value-param) [Ln 130, Col 53]
The parameter 's2' is copied for each invocation but only used as a const reference; consider making it a const reference clang-tidy(performance-unnecessary-value-param) [Ln 130, Col 63]
2 adjacent parameters of 'warn' of similar type ('const std::string &') are easily swapped by mistake clang-tidy(bugprone-easily-swappable-parameters) [Ln 137, Col 20]
Logger.cpp[Ln 137, Col 34]: The first parameter in the range is 'methodName'
Logger.cpp[Ln 137, Col 60]: The last parameter in the range is 's'
2 adjacent parameters of 'error' of similar type ('const std::string &') are easily swapped by mistake clang-tidy(bugprone-easily-swappable-parameters) [Ln 144, Col 21]
Logger.cpp[Ln 144, Col 35]: The first parameter in the range is 'methodName'
Logger.cpp[Ln 144, Col 61]: The last parameter in the range is 's'
Logger.h restful-with-billions • include 4
Variable '_dump' is non-const and globally accessible, consider making it const clang-tidy(cppcoreguidelines-avoid-non-const-global-variables) [Ln 17, Col 14]

Injection

Variable <…> is non-const and globally accessible, consider making it const

```
void Restful::start() {
  http::host host;
  host.setup(App::getConfig().at("httpServer"));

  std::ostringstream os;
  os << "Starting server on " << host.ip << ":" << host.port << std::endl;

  std::thread(&http::listener::run, std::move(host)).detach();

  Logger::getLogger()->save(::logline(static_cast<const char *>(__func__)),
  "Starting server " + VERSION + "....", IS_MAIN);
}
```

Bridging with solid types

```
{
"server": {
   "ip": "*",
   "appPort": 9100
},
"timers": {
   "monitorTime": 30,
   "downTime": 1,
   "removeTime": 1
}
}
```

```
Cortex::Cortex( const json& config, zmq::context_t& context )
: config( config ), context( context ),
monitorTime( config[ "timers" ][ "monitorTime" ] ),
downTime( config[ "timers" ][ "downTime" ] ),
removeTime( config[ "timers" ][ "removeTime" ] ) {

thread( &Cortex::monitor, this ).detach( );
thread( &Cortex::publisher, this ).detach( );

logger = Logger::getLogger( );

this->ctxId = AppIdentity( AppType::mcp,
config[ "server" ][ "ip" ].get< string >( ),
config[ "server" ][ "appPort" ].get< ushort >( ) );
}
```

Better with solid types

```
struct Host {
  std::string ip;
  int port;
};

from_json(const json& j, Host &h) {
  h.ip = j.value("ip", "*");
  h.port = j.value("port", 80);
}

struct Timers {
  uint monitorTime;
  uint downTime;
  uint removeTime;
};

from_json(const json &j, Timers &t) {
  t.monitorTime = j.value("monitorTime", 60);
  t.downTime = j.value("downTime", 1);
  t.removeTime = j.value("removeTime", 1);
}
```

```
struct CortexConfig {
  Timers timer;
  Host host;
};

from_json(const json& j, CortexConfig &cf) {
  cf.host = j.value("host", Host{});
  cf.timer = j.value("timers", Timers{});
}

 Cortex::Cortex( const CortexConfig config, zmq::context_t&
 context )
 : config( CortexConfig ), context( context ) {
 thread( &Cortex::monitor, this ).detach( );
 thread( &Cortex::publisher, this ).detach( );

 logger = Logger::getLogger( );

 this->ctxId = AppIdentity( AppType::mcp, host );
 }
```

Findings

- Speed and Performance
  - > $100 billion annually
  - > 700 million transactions
- With lots of room to grow.
- Easy to Integrate.

Conclusion: What does it mean?

- Serving or Consuming JSON and REST interfaces in C++ is simple.
- Cross platform and efficient.
- JSON allows easy sharing with other applications.
- Useful design does not paint yourself into a corner.

https://github.com/kevinbcarpenter/restful-with-billions

Thank You

https://github.com/kevinbcarpenter/restful-with-billions