Example 3

Essential Packages on Ubuntu

1. apt-get update

```
apt-get update
```

2. apt-get install -y sudo vim wget unzip g++ cmake curl pkg-config libssl-dev libsasl2-dev git python3 nano

```
apt-get install -y sudo vim wget unzip g++ cmake curl pkg-config libssl-dev
libsasl2-dev git python3 nano
```

3. mkdir restapicpp

```
mkdir restapicpp
```

4. cd restapicpp

```
cd restapicpp
```

MongoDB C Driver

5. wget https://github.com/mongodb/mongo-c-driver/releases/download/1.24.4/mongo-c-driver-1.24.4.tar.gz

```
wget https://github.com/mongodb/mongo-c-driver/releases/download/1.24.4/mongo-c-
driver-1.24.4.tar.gz
```

6. tar -xzvf mongo-c-driver-1.24.4.tar.gz

```
tar -xzvf mongo-c-driver-1.24.4.tar.gz
```

7. cd mongo-c-driver-1.24.4/build

```
cd mongo-c-driver-1.24.4/build
```

8. cmake ..

```
cmake ..
```

9. cmake --build . --config RelWithDebInfo --target install

```
cmake --build . --config RelWithDebInfo --target install
```

10. cd ../..

```
cd ../..
```

MongoDB C++ Driver

11. wget https://github.com/mongodb/mongo-cxx-driver/releases/download/r3.7.0/mongo-cxx-driver-r3.7.0.tar.gz

```
wget https://github.com/mongodb/mongo-cxx-driver/releases/download/r3.7.0/mongo-cxx-driver-r3.7.0.tar.gz
```

12. tar -xzvf mongo-cxx-driver-r3.7.0.tar.gz

```
tar -xzvf mongo-cxx-driver-r3.7.0.tar.gz
```

13. cd mongo-cxx-driver-r3.7.0/build

```
cd mongo-cxx-driver-r3.7.0/build
```

14. cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local ..

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release -
DCMAKE_INSTALL_PREFIX=/usr/local ..
```

15. cmake --build . --target install

```
cmake --build . --target install
```

16. cd ../..

```
cd ../..
```

Extract Crow Framework

17. wget https://github.com/CrowCpp/Crow/releases/download/v1.0%2B5/crow-v1.0+5.tar.gz

```
wget https://github.com/CrowCpp/Crow/releases/download/v1.0%2B5/crow-v1.0+5.tar.gz
```

18. mkdir crow

```
mkdir crow
```

19. tar xvfz crow-v1.0+5.tar.gz -C crow --strip-components=1

```
tar xvfz crow-v1.0+5.tar.gz -C crow --strip-components=1
```

Extract Boost Libraries

20. wget https://boostorg.jfrog.io/artifactory/main/release/1.83.0/source/boost_1_83_0.tar.gz

```
wget
https://boostorg.jfrog.io/artifactory/main/release/1.83.0/source/boost_1_83_0.tar.
gz
```

21. tar -xzvf boost_1_83_0.tar.gz

```
tar -xzvf boost_1_83_0.tar.gz
```

22. nano main.cpp

```
nano main.cpp
```

```
#include "Methods.h"

// ****************************************** Main
******************************************
int main()
{
    crow::SimpleApp app; //define your crow application
    set_global_base("."); //search for the files in current dir.
    mongocxx::instance inst{};
    string mongoConnect = std::string("your_mongodburi");
    mongocxx::client conn{ mongocxx::uri{mongoConnect} };
    auto collection = conn["TodoRecords"]["TodoCollection"];//get collection from
database

    //API endpoint to read all todos
    CROW_ROUTE(app, "/api/v1/todos").methods(HTTPMethod::GET)
        ([&collection](const request& req) {
        mongocxx::options::find opts;
        auto docs = collection.find({}, opts);
        vector<crow::json::rvalue> todo;

        for (auto doc : docs) {
            todo.push_back(json::load(bsoncxx::to_json(doc)));
        }
        crow::json::wvalue dto;
        dto["todos"] = todo;
        return crow::response{ dto };
            });

    //API endpoint to insert todo from the given json body
    CROW_ROUTE(app, "/api/v1/todos").methods(HTTPMethod::POST)
        ([&collection](const request& req) {
        crow::json::rvalue request_body = json::load(req.body);

        // List of required keys
        std::vector<std::string> required_keys = { "Id", "firstName", "lastName",
"emailId", "location" };

        // Check if all required keys exist in the request body
        for (const auto& key : required_keys) {
            if (!request_body.has(key)) {
                return crow::response(400, "Required key '" + key + "' missing in
request body");
            }
        }

        // Check if the ID is already in the database
        bool id_already_present = findTodoRecord(collection,
std::string(request_body["Id"]));

        if (!id_already_present) {
            // ID is not present, so insert the new record
            insertTodo(collection, createTodo({
```

```
                {"Id", std::string(request_body["Id"])},
                {"firstName", std::string(request_body["firstName"])},
                {"lastName", std::string(request_body["lastName"])},
                {"emailId", std::string(request_body["emailId"])},
                {"location", std::string(request_body["location"])},
                }));
            return crow::response(200, "Todo Added Successfully!!");
        }
        else {
            // ID is already present
            return crow::response(400, "ID already present in the database");
        }
            });

    //API endpoint to update document based on the given id in the JSON body
        CROW_ROUTE(app, "/api/v1/todos").methods(HTTPMethod::PUT)
        ([&collection](const request& req) {
        crow::json::rvalue request_body = json::load(req.body);

        if (!request_body.has("Id")) {
            return crow::response(400, "Required key 'Id' missing in request
body");
        }

        std::string id = std::string(request_body["Id"]);
        bool id_already_present = findTodoRecord(collection, id);

        if (id_already_present) {
            // ID is present, so update the record
             vector<pair<string, string>> updates;
             for (auto& item : request_body) {
                 string key = item.key();
                 if (key != "Id") {
                     try {
                         string value = item.s();
                         updates.push_back({ key, value });
                     }
                     catch (const std::runtime_error&) {
                         // Not a string, ignore or handle accordingly
                     }
                 }
             }
            if (updateTodo(collection, "Id", id, updates)) {
                return crow::response(200, "Todo Updated Successfully!!");
            }
            else {
                return crow::response(500, "Failed to update Todo");
            }
        }
        else {
            return crow::response(400, "ID not found in the database");
        }
            });
```

```
        #undef DELETE

    // endpoint to delete a document based on the given id in the JSON body
        CROW_ROUTE(app, "/api/v1/todos").methods(HTTPMethod::DELETE)
        ([&collection](const request& req) {
        crow::json::rvalue request_body = json::load(req.body);

        if (!request_body.has("Id")) {
            return crow::response(400, "Required key 'Id' missing in request
body");
        }

        std::string id = std::string(request_body["Id"]);
        if (deleteTodo(collection, "Id", id)) {
            return crow::response(200, "Todo Deleted Successfully!!");
        }
        else {
            return crow::response(400, "Failed to delete Todo with given ID");
        }
            });

        //API endpoint to read a specific document by Id
        CROW_ROUTE(app, "/api/v1/todos/<string>").methods(HTTPMethod::GET)
            ([&collection](const string& id) {
            // Create the query filter based on the provided Id
            auto filter = bsoncxx::builder::stream::document{} << "Id" << id <<
bsoncxx::builder::stream::finalize;

            auto maybe_result = collection.find_one(filter.view());

            if (maybe_result) {
                auto doc = maybe_result->view();
                crow::json::wvalue dto;
                dto["todo"] = json::load(bsoncxx::to_json(doc));
                return crow::response{ dto };
            }
            else {
                return crow::response(404, "Todo not found");
            }
                });


    //set the port, set the app to run on multiple threads, and run the app
    app.bindaddr("0.0.0.0").port(8080).multithreaded().run();

}
```

23. nano Methods.h

```
nano Methods.h
```

```
#pragma once
#include <mongocxx/client.hpp>
#include <bsoncxx/builder/stream/document.hpp>
#include <bsoncxx/json.hpp>
#include <mongocxx/uri.hpp>
#include <mongocxx/instance.hpp>
#include <algorithm>
#include <iostream>
#include <vector>
#include "crow.h"
using namespace std;
using namespace crow;
using namespace crow::mustache;
using bsoncxx::builder::basic::kvp;
using bsoncxx::builder::basic::make_document;

// Create a todo from the given key-value pairs.
bsoncxx::document::value createTodo(const vector<pair<string, string>>& keyValues)
{
    bsoncxx::builder::stream::document document{};
    for (auto& keyValue : keyValues)
    {
        document << keyValue.first << keyValue.second;
    }
    return document << bsoncxx::builder::stream::finalize;
}

// Add the todo to the given collection.
void insertTodo(mongocxx::collection& collection, const bsoncxx::document::value&
document)
{
    collection.insert_one(document.view());
}

// Find a todo from the given key-value pairs and return true if found.
bool findTodo(mongocxx::collection& collection, const string& key, const string&
value)
{
    // Create the query filter
    auto filter = bsoncxx::builder::stream::document{} << key << value <<
bsoncxx::builder::stream::finalize;
    //Add query filter argument in find
    auto cursor = collection.find({ filter });
    auto count = std::distance(cursor.begin(), cursor.end());
    if (count != 0L) {
        return true;
    }
    return false;
}

//Pass the given collection and key-value pairs.
bool findTodoRecord(mongocxx::collection& collection, const string& id)
```

```
{
    return findTodo(collection, "Id", id);
}

// Update a todo in the given collection based on a specific key-value pair with
new key-value pairs.
bool updateTodo(mongocxx::collection& collection,
    const string& key, const string& value,
    const vector<pair<string, string>>& newKeyValues)
{
    // Create the query filter to find the document to update
    auto filter = bsoncxx::builder::stream::document{} << key << value <<
bsoncxx::builder::stream::finalize;

    // Create the new values for the document
    bsoncxx::builder::stream::document updated_document{};
    for (auto& keyValue : newKeyValues)
    {
        updated_document << keyValue.first << keyValue.second;
    }

    auto update_doc = bsoncxx::builder::stream::document{} << "$set" <<
updated_document << bsoncxx::builder::stream::finalize;

    // Update the document
    auto result = collection.update_one(filter.view(), update_doc.view());

    // Return true if at least one document was modified
    return result && result->modified_count() > 0;
}

// Delete a todo from the given collection based on a specific key-value pair.
bool deleteTodo(mongocxx::collection& collection, const string& key, const string&
value)
{
    // Create the query filter to find the document to delete
    auto filter = bsoncxx::builder::stream::document{} << key << value <<
bsoncxx::builder::stream::finalize;

    // Delete the document
    auto result = collection.delete_one(filter.view());

    // Return true if at least one document was deleted
    return result && result->deleted_count() > 0;
}
```

24. nano CMakeLists.txt

```
nano CMakeLists.txt
```

```
cmake_minimum_required(VERSION 3.15)
project(restapicpp)

# Define the include directories
set(INCLUDE_PATHS ./boost_1_83_0 ./crow/include)

# Add the executable target
add_executable(restapicpp main.cpp)

# Include the defined paths
target_include_directories(restapicpp PUBLIC ${INCLUDE_PATHS})

# MongoDB C++ driver includes and links
# The Dockerfile does not provide the exact paths, but often, the drivers get
installed to /usr/local
set(MONGOCXX_LIBS /usr/local/lib)
set(MONGOCXX_INCLUDE /usr/local/include/mongocxx/v_noabi)
set(BSONCXX_INCLUDE /usr/local/include/bsoncxx/v_noabi)

target_include_directories(restapicpp PRIVATE ${MONGOCXX_INCLUDE}
${BSONCXX_INCLUDE})
target_link_libraries(restapicpp PRIVATE
    ${MONGOCXX_LIBS}/libmongocxx.so
    ${MONGOCXX_LIBS}/libbsoncxx.so
)

# Specify the C++ standard
set_target_properties(restapicpp PROPERTIES
    CXX_STANDARD 17
    CXX_STANDARD_REQUIRED TRUE
)
```

Build the Application

25. mkdir builds

```
mkdir builds
```

26. cd build/

```
cd build/
```

27. cmake ..

```
cmake ..
```

28. make

```
make
```

Run the Application

29. ./restapicpp

```
./restapicpp &
```

Once executed, it will start a web server on a specified port 8080

POST Request:

Posting a New Todo Item to the API

```
curl --location --request POST 'http://localhost:8080/api/v1/todos' \
--header 'Content-Type: application/json' \
--data-raw '{
    "Id": "1",
    "firstName": "firstName",
    "lastName": "lastName",
    "emailId": "email@gmail.com",
    "location": "location"
}
```

Expected Output:

```
Todo Added Successfully!!
```

GET Request:

Fetching All Todo Items from the API.

```
curl --location --request GET 'http://localhost:8080/api/v1/todos'
```

Expected Output:

```
{"todos":
[{"location":"location","lastName":"lastName","emailId":"email@gmail.com","firstNa
me":"firstName","Id":"1","_id":{"$oid":"6514fb50f82cdde1bd0dfe11"}}]}
```

PUT Request

```
curl --location --request PUT 'http://localhost:8080/api/v1/todos' \
--header 'Content-Type: application/json' \
--data '{
    "Id": "1",
    "location": "Chennai"
}'
```

Expected Output:

```
Todo Updated Successfully!!
```

DELETE Request

```
curl --location --request DELETE 'http://localhost:8080/api/v1/todos' \
--header 'Content-Type: application/json' \
--data '{
    "Id":"1"
}'
```

Expected Output:

```
Todo Deleted Successfully!!
```

GET BY ID Request

```
curl --location 'http://localhost:8080/api/v1/todos/1'
```

Expected Output:


```
{"todo":
{"location":"Chennai","lastName":"lastName","emailId":"email@gmail.com","firstName
":"firstName","Id":"1","_id":{"$oid":"6515254668b0190b790b0fd1"}}}
```