

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп'ютерних систем

**КУРСОВА РОБОТА**  
*з дисципліни "Основи програмування"*

Виконав: Кравчук Аркадій Андрійович  
Група: КП-71

Допущено до захисту

---

2 семестр 2017/2018

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«Київський політехнічний інститут імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра програмного забезпечення комп'ютерних систем

Узгоджено

ЗАХИЩЕНА " \_\_ " \_\_\_\_\_ 2018р.

Керівник роботи

з оцінкою \_\_\_\_\_

\_\_\_\_\_/Гадиняк Р.А./

\_\_\_\_\_/Гадиняк Р.А./

**Розробка мікрофреймворка для web-додатків**

Виконавець роботи

Кравчук Аркадій Андрійович

\_\_\_\_\_ 2018р.

## Вступний опис системи

Розроблена система представляє собою web-фреймворк, за допомогою якого було створено демо-сайт логістичної компанії, а також графічний клієнт, користуючись яким можна посилати запити на розроблений додаток та переглядати відповіді у загальній формі, а також у форматі JSON.

Так як сьогодні миттєве отримання будь-якої інформації у будь-якій точці світу стало невід'ємною рисою сучасності, тому обробка, отримання, пересилання даних через Інтернет є важливим та актуальним напрямком розробки. Саме завдяки використанню розробленого мікрофреймворка можна створювати екземпляри веб-додатків, кожен з яких є самостійним веб-сервером і може легко налаштовуватись для відображення результатів за допомогою обробників, модулів розширень тощо. Завдяки тому, що розроблений комплекс програм був розроблений на компільованій мові програмування C++, веб-фреймворк відзначається високою швидкістю в порівнянні з аналогами, написаних на таких інтерпретованих мовах як PHP, Perl, Ruby. Тому розроблене програмне забезпечення є дійсно актуальним.

За допомогою документації (див. Додатки) можна швидко “підняти” свій сервер та наповнити його функціоналом. Завдяки розробленим розширенням можна запитувати інформацію з бази даних, генерувати будь-які сторінки за допомогою рендеру заданого шаблону із контекстом значень через спеціальне розширення. Мікрофреймворк підтримує більшість стандартів та можливостей HTTP: Cookies, Redirects, парсинг словника параметрів запиту URI, обробка даних від форм application/x-www-form-urlencoded та багато іншого. Користувачем можуть створюватись однорангові обробники, обробники заданих URI запитів тощо. Також є можливість швидко підключати будь-які файли з файлової системи сервера та поширювати їх у мережі.

На основі розроблених модулів користувач веб-фреймворку може створювати власні модулі-розширення та обробники, наслідуючись від базових класів. Однією із головних особливостей є те, що веб-сервер можна інтегрувати з будь-яким програмним забезпеченням за допомогою

поширеного формату даних - JSON. Таким чином можна створювати власні APIs для взаємодії з зовнішніми додатками.

## Інструменти розробки

Розробка веб-сервера велась в інтегрованому середовищі розробки “CLion” на мові програмування C++. При написанні сервера використовувались наступні бібліотеки:

- [nlohmann\\_json](#) - використовувалась для формування та маніпуляцій з json об'єктами та масивами для подальшої серіалізації у http відповідь; для парсингу json строк із http запитів.
- [mstch](#) - logic-less шаблонізатор сторінок mustache, який дозволяє формувати веб-сторінки відповіді сервера.
- [sqlite3](#) - використовувалась для здійснення підключень до бази даних, та виконання SQL запитів до неї

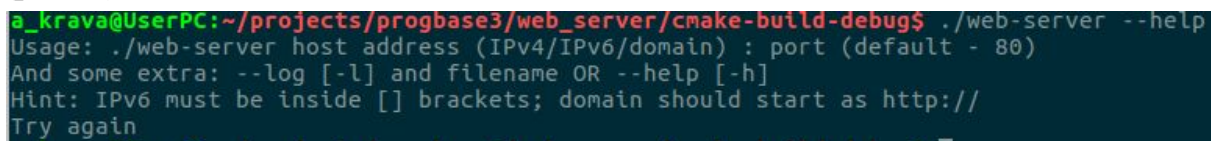
Розробка клієнта велась на мові програмування C++ на фреймворці Qt 5.9.3, в інтегрованому середовищі розробки Qt Creator 4.4.1. При написанні клієнта використовувались наступні стандартні бібліотеки Qt:

- *QTcpSocket* - користування мережевими сокетами для взаємодії з сервером
- *QHostAddress* - для налаштування IP адреси
- *QSyntaxHighlighter* - для підсвічування синтаксису JSON
- *QTextCharFormat* - для форматування інформації, стилю символів
- *QRegExp* - використання регулярних виразів для валідації введеної адреси та підсвітки синтаксису
- *QTextDocument* - для доступу до форматowanego тексту у віджетах
- *QMainWindow* - створення графічного вікна
- *QApplication* - забезпечення подійно-орієнтованої системи
- *QMessageBox* - створення інформаційних діалогових вікон

## Реалізований функціонал системи

- Фреймворк дозволяє створювати екземпляри веб-додатків та запускати їх на певній IP-адресі та порті як веб-сервери.

Було розроблено спеціальний модуль InitParams, який парсить аргументи командної строки, та відповідно заданому формату зчитує адресу та порт для запуску сервера. Також є можливість задати назву для файлу, в який будуть логуватися всі критичні події (за допомогою модуля LogManager), а також дізнатися формат введення адреси сервера. Кожен екземпляр є самостійною консольною програмою, приклад запуску наведено нижче: (рис. 1)



```
a_krava@UserPC:~/projects/progbase3/web_server/cmake-build-debug$ ./web-server --help
Usage: ./web-server host address (IPv4/IPv6/domain) : port (default - 80)
And some extra: --log [-l] and filename OR --help [-h]
Hint: IPv6 must be inside [] brackets; domain should start as http://
Try again
```

Рис. 1

Додано підтримку протоколу IPv6, а також запуск web-сервера заданням доменної адреси. Слід відзначити, що введені дані перевіряються на валідність, тому запуск програми завчасно завершується. Лістинг коду перевірки на валідність ір адреси:

init\_params.cpp

```
static bool isValidIP(string & host, bool * IPv6) {
    stringstream IP(host);
    string block;
    if (host.substr(0, 4).find('.') != string::npos) {
        for (int i = 0; i < 4; i++) {
            if (!getline(IP, block, '.') || !isValidIPv4Block(block)) return false;
        }
        *IPv6 = false;
        return IP.eof();
    } else if (host.substr(0, 5).find(':') != string::npos) {
        for (int i = 0; i < 8; i++) {
            if (!getline(IP, block, ':') || !isValidIPv6Block(block)) return false;
        }
        *IPv6 = true;
        return IP.eof();
    }
    return false;
}
```

```
// https://en.wikipedia.org/wiki/IPv4#Addressing
static bool isValidIPv4Block(string & block) {
    int num = 0;
    if (!block.empty() && block.size() <= 3) {
        for (size_t i = 0; i < block.size(); i++) {
            char c = block[i];
            if (!isdigit(c) || (i == 0 && c == '0' && block.size() > 1)) {
                return false;
            }
        }
    }
    return true;
}
```

```

        } else {
            num *= 10;
            num += c - '0';
        }
    }
    return num <= 255;
}
return false;
}

// https://en.wikipedia.org/wiki/IPv6#Addressing
static bool isValidIPv6Block(string & block) {
    if (!block.empty() && block.size() <= 4) {
        for (char & c : block) {
            if (!isdigit(c)) return false;
        }
        return true;
    }
    return false;
}
}
...

```

Модуль Socket реалізує створення блокуючого сокету на визначеній адресі, а також отримання та відправлення даних до клієнта.

- Можна налаштувати *Application* за допомогою обробника запитів (*Handler*) загального призначення, який оброблятиме будь-які *HTTP*-запити до додатку. Також можна задати декілька однорангових обробників, тоді вони сформують ланцюжок обробників запити.

Модуль *Application* реалізує головні функції програми. Він має списки, хеш-таблиці посилань на додані обробники. Кожен обробник є екземпляром модуля *Handler*, в якому є метод *exec*. Останній визначає роботу обробника для кожного запиту. Після запуску модуля *Application* методом *run* запускається вічний цикл, в якому спершу слухається запит від клієнта, а в останню чергу відправляється сформована обробниками відповідь клієнту. Слід зазначити, що серіалізація/десеріалізація відбувається в окремому модулі *ParserHTTP*. Обробники загального призначення додаються у вектор, який міститься у *Application*. Після прийому запиту від клієнта, в циклі усі додані загальні обробники виконують метод *exec*.

- Кожен *Handler* викликається із посиланням на додаток, у кому він зареєстрований (*App*), із контекстом запиту (*Context*) та посиланням на наступний *Handler* (*Next*) із ланцюжка налаштованих обробників.

У модулі Application було реалізовано метод addHandler який додає до внутрішнього списку даних обробник. Перед виконанням кожного обробника йому встановлюється в поле Context екземпляр поточного об'єкту запиту та оброблений попередніми обробниками (або пустий) екземпляр об'єкту відповіді. Context є обгорткою для об'єктів запиту/відповіді.

- *Context у собі містить розібрані дані із HTTP-запиту (дані першого рядка, аргументи URL, словник заголовочних полів та тіло запиту) у вигляді екземплярів відповідних класів та методи для встановлення даних HTTP-відповіді (дані першого рядка, словник заголовочних полів та тіло відповіді).*

Об'єкт модуля Context складається з об'єктів модулів Response та Request відповідно. Модуль Request містить у собі наступні поля: Method - значення з переліку всіх можливих методів, об'єкт модуля URI, Version - певне значення з переліку версій, об'єкт модуля Headers та об'єкт модуля MessageBody. Модуль Response містить у собі наступні поля: Version - версію http, status - статус виконання запиту, Headers - заголовочні поля відповіді, MessageBody - тіло відповіді. Кожен внутрішній модуль парсить/серіалізує свою частину http запиту/відповіді. Причому кожен модуль надає функції для опрацювання та взаємодії зі своїми даними обробникам. Важливою частиною формування http запитів та відповідей є кодування та декодування даних, коли не ASCII символи набувають HEX вигляду. Наведемо лістинги коду цих функцій:

parser\_http.cpp

```
...
string ParserHTTP::urlEncode(const string &value) {
    ostringstream escaped;
    escaped.fill('0');
    escaped << hex;
    for (auto & c : value) {
        if (isalnum(c) || containsArr(reservedSymbols, sizeof(reservedSymbols) / sizeof(char), c)) {
            escaped << c;
        } else {
            escaped << uppercase;
            escaped << '%' << setw(2) << int((unsigned char) c);
            escaped << nouppercase;
        }
    }
    return escaped.str();
}

string ParserHTTP::urlDecode(const string & value) {
```



```

ostream result;
for (size_t i = 0; i < value.length(); i++) {
    string::value_type c = value[i];
    if (c == '+') {
        result << ' ';
    } else if (isalnum(c) || containsArr(reservedSymbols, sizeof(reservedSymbols) / sizeof(char),
c)) {
        result << c;
    } else if (c == '%' && i + 2 < value.length()) {
        string hex = string();
        if (!isxdigit(value[i + 1]) || !isxdigit(value[i + 2])) {
            return value;
        }
        hex += value[++i];
        hex += value[++i];
        auto cur = (unsigned char) stoi(hex, nullptr, 16);
        result << cur;
    } else {
        return value;
    }
}
return result.str();
}

```

- Можна налаштувати обробники для визначених URL-шляхів (Route) та визначених HTTP-методів із внесенням параметрів у ці шляхи за допомогою простого формату, та із автоматичним парсингом всіх таких параметрів у контекст запиту у вигляді словника (Params).

Для обробників було створено додаткове поле для url шляхів та методу, а також додано новий конструктор для цих даних. При додаванні до Application перевіряється, чи є url шлях у даного обробника. Якщо так, тоді він додається до хеш таблиці за ключем url шляху, в протилежному випадку він є загальним обробником. У модулі Application після виконання всіх загальних обробників, шукається у хеш таблиці обробник за поточною url адресою запиту. Якщо відповідний елемент буде знайдено, а також метод запиту та встановлений співпадуть, тоді знайдений обробник виконається. Параметри, задані в url шляху парсяться модулем URI у внутрішній словник params. Лістинг даного методу:

uri.cpp

```

...
void URI::setParamsAndUri(string & uri) {
    string path;
    size_t startParamsPos = uri.find('?');
    if (startParamsPos == string::npos) {
        path = ParserHTTP::urlDecode(uri);
    } else {
        path = uri.substr(0, startParamsPos);
        path = ParserHTTP::urlDecode(path);
        size_t startKeyPos = startParamsPos + 1;
        while (startKeyPos < uri.length()) {

```

```

        size_t endKeyPos = uri.find('=', startKeyPos);
        if (endKeyPos == string::npos) break;
        string key = uri.substr(startKeyPos, endKeyPos - startKeyPos);
        size_t endValuePos = uri.find('&', endKeyPos + 1);
        if (endValuePos == string::npos) {
            endValuePos = uri.length();
        }
        string value = uri.substr(endKeyPos + 1, endValuePos - endKeyPos - 1);
        key = ParserHTTP::urlDecode(key);
        value = ParserHTTP::urlDecode(value);
        params.insert({key, value});
        startKeyPos = endValuePos + 1;
    }
}
this->uri = path;
}
...

```

- *Context* дозволяє виконати перенаправлення (*redirect*) на іншу веб-сторінку.

Для кожного об'єкта модуля *Context* було створено три методи для встановлення перенаправлень: *setPermanentlyRedirect* (для постійних перенаправлень), *setTemporaryRedirect* (для тимчасових перенаправлень), *setRedirect* (для перенаправлень з визначеним http кодом статусу). Усі ці методи формують спеціальні заголовки відповіді в об'єкті *Headers*. Також розроблено модуль *RedirectResponse*, який вже формує готову http відповідь для перенаправлення за визначеною адресою. Для *Application* було розроблено методи для додавання перенаправлень у простому форматі. Під час кожного запиту здійснюється пошук, чи було додано відповідний *Redirect*.

- *Context* містить словник розширення, що дозволяє обробникам додавати у нього власні дані та функції. Це дозволить створювати модулі-посередники (*Middleware*), що розширюватиме базовий функціонал *Application*.

Кожен об'єкт модуля *Middleware* має посилання на поточний екземпляр http запиту, має внутрішній словник, та має доступ до формування http відповіді. Причому, модулі *Application* та *Context* мають посилання до всіх доданих розширень *Middleware*. Таким чином під час кожного оброблення http запитів відбувається автоматичне виконання налаштованих попередньо дій для кожного розширення (якщо таке можна виконати). Кожен обробник має доступ до розширень, знаючи текстовий

ідентифікатор. Таким чином обробнику можна переглянути результати автоматичного виконання розширень, а також скористатись вбудованими методами для формування http відповіді.

- *Створити Middleware, що автоматично парсить JSON у тілі запиту, якщо встановлене відповідне значення у заголовочному полі Content-Type. Також цей модуль дозволяє додати у відповідь сервера функцію, що автоматично із JSON-об'єкта формує тіло відповіді у форматі JSON.*

Було розроблено JsonMiddleware, для якого розроблено відповідні поля для запису json запиту та відповіді. Якщо тип запиту визначений як json, тоді відбувається автоматичний парсинг даних. Кожному обробнику доступні розібрані дані, а також є можливість формування json об'єктів та метод для встановлення даних у форматі json у http відповідь.

- *Створити Middleware для HTTP Cookie, що парсить із відповідного HTTP заголовку словник Cookie та дозволяє встановити нові Cookie у відповідь сервера.*

Було розроблено модуль розширення CookieMiddleware для парсингу Cookie з http запиту та їх встановлення у http відповідь. Кожен обробник може дізнатися про надіслані Cookie, а також встановити клієнту власні. Для Cookie було розроблено модуль CookieEntity, який дозволяє детально налаштувати кожен Cookie. Наведемо лістинг коду серіалізації CookieEntity:

cookie\_entity.cpp

```
...
std::string CookieEntity::toString() {
    std::string result = ParserHTTP::urlEncode(value);
    if (expires != -1) {
        result += "; Expires=";
        tm * ltm = std::localtime(&expires);
        char buff[100];
        if (strftime(buff, sizeof(buff), "%a, %d %b %Y %H:%M:%S GMT", ltm) > 0) {
            result += buff;
        }
    }
    if (maxAge_sec != std::string::npos) {
        result += "; Max-Age=" + maxAge_sec;
    }
    if (!domain.empty()) {
        result += "; Domain=" + ParserHTTP::urlEncode(domain);
    }
}
```

```
if (!path.empty()) {  
    result += "; Path=" + ParserHTTP::urlEncode(path);  
}  
if (httpOnly) {  
    result += "; HttpOnly";  
}  
return result;  
}
```

...

- Створити *Middleware* для запиту від форм, що містять тіло у форматі *application/x-www-form-urlencoded*. На основі даних такого тіла формується словник даних, що були внесені користувачем у формі веб-сторінки.

Було розроблено модуль *FormMiddleware*, який автоматично парсить в словник дані з тіла запиту, якщо було встановлено відповідний тип контенту. Кожен обробник має доступ до розібраних даних з форми.

- Створити *Middleware*, що дозволить формувати веб-сторінки відповіді сервера використовуючи простий *logic-less* шаблонізатор.

Модуль *HtmlMiddleware* використовує шаблонізатор *mustache*, який дозволяє з заданого шаблону та сформованого контексту формувати *html* сторінки та вставляти результат у тіло *http* відповіді.

- Необхідно створити демонстраційний графічний клієнт, що дозволить виконувати *HTTP* запити до веб-сервера та відображати відповіді.

За допомогою фреймворка *Qt* було розроблено графічну віконну програму, яка дозволяє відправляти запити до веб-сервера, із введених користувачем заголовків та тіла запиту, і переглядати отримані відповіді. Також є можливість надсилати *json* запити і переглядати *json* відповідь: (Рис. 2)

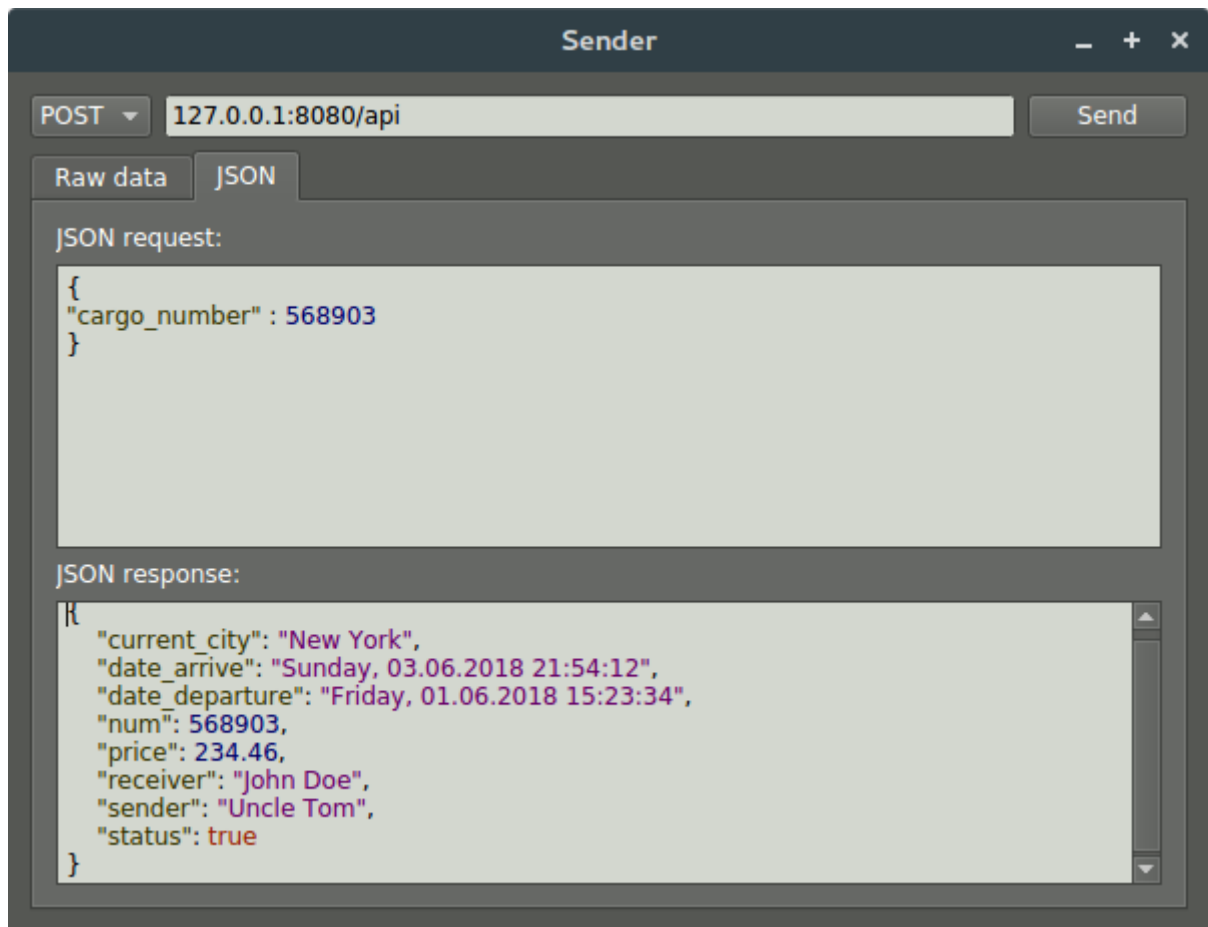


Рис. 2

Було розроблено підсвітку синтаксису json, код для реалізації якої був взятий з [github](#).

- *Необхідно створити демонстраційний веб-сервер для логістичної компанії, що використовує всі розроблені можливості і дозволяє обробляти клієнтські запити, підключатись до простої бази даних для отримання відповідних даних та генерувати клієнту за допомогою шаблонізатора веб-сторінки із результатами.*

На основі написаного фреймворку було створено веб-сервер логістичної компанії згідно до зазначеного функціоналу.

- *Розроблено наступний функціонал для демонстраційного веб-серверу:*
  - *відстеження відправленого вантажу користувачем за номером квитанції. Для кожної квитанції зберігатиметься інформація про відправника, одержувача, дату прибуття та відправлення,*

поточне місцезнаходження та вартість доставки у БД. Ці дані будуть надіслані та відображені на веб-сторінці.

- розрахунок приблизної вартості доставки вантажу із зазначеними даними: габаритами, вагою, містом відправника та одержувача. На сервер буде здійснено відповідний запит, внаслідок чого користувач отримає бажану інформацію.
- розрахунок приблизного терміну доставки на основі зазначеної дати відправлення, міста відправника та одержувача.
- отримання інформації (графіку роботи, адреси) про відділення логістичної компанії у зазначеній(-ому) області(місті).

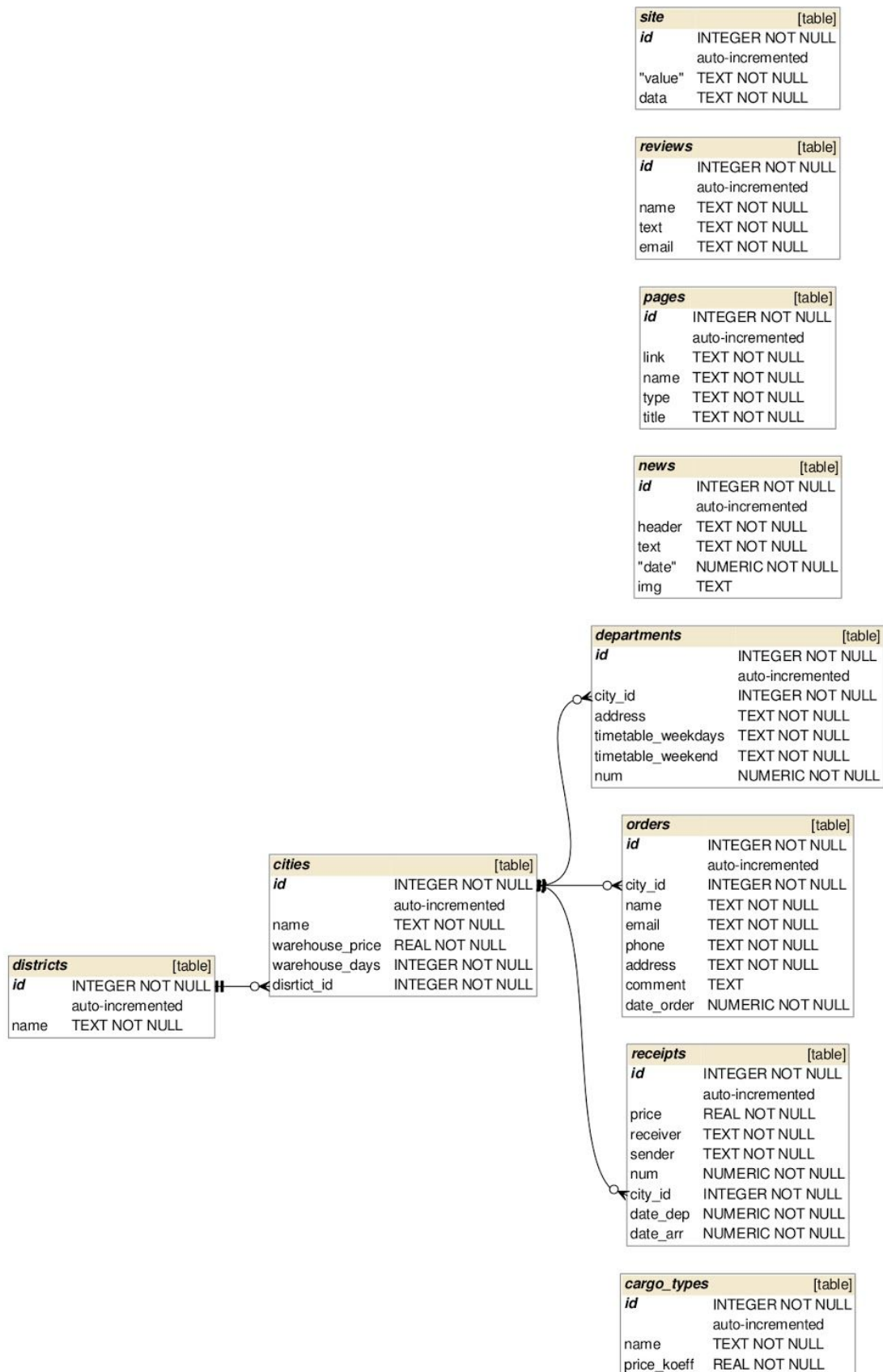
Приклад запиту до БД:

main.cpp

```
...  
"SELECT departments.num, cities.name, districts.name, departments.address,  
departments.timetable_weekdays, departments.timetable_weekend FROM departments INNER JOIN  
cities ON departments.city_id = cities.id INNER JOIN districts ON cities.distrtict_id =  
districts.id WHERE cities.id = ? OR districts.id = ?"  
...
```

- можливість виклику кур'єра з сайту, заповнивши форму (вказавши ПІБ, адресу), для відправлення/отримання вантажу.
- Відображення загальної інформації про компанію, новини у відповідних розділах, а також можливість відправити відгук (т.з. *feedback*)

## Організація даних



На вищенаведеній схемі наведено структуру та зв'язки в базі даних SQLite, яка використовується для зберігання та відображення даних на демонстраційному веб-сервері.

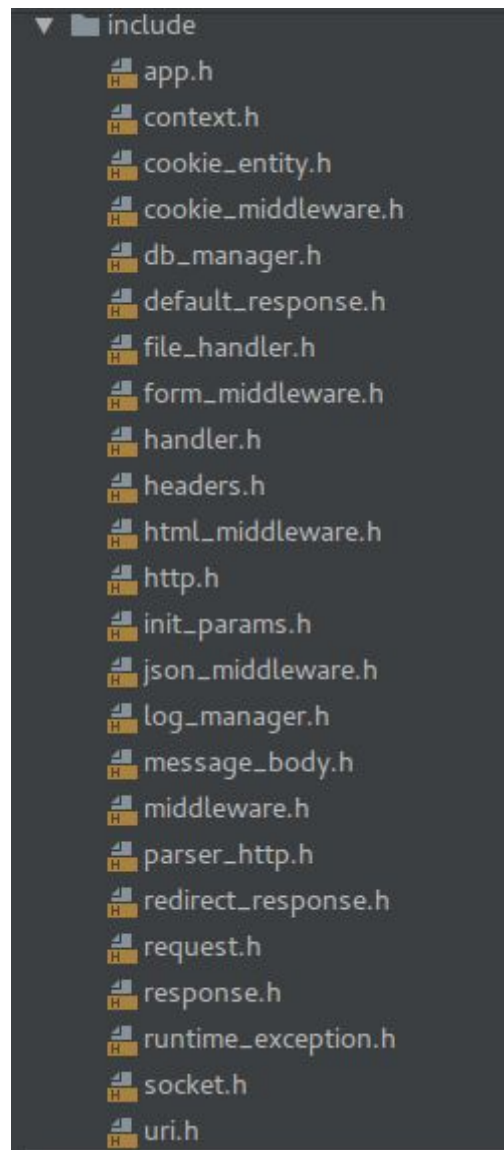
Також на демонстраційному веб-сервері розроблено отримання даних про відправлення за допомогою JSON запиту. Для цього на сторінку */api* відправте json об'єкт із полем *cargo\_number* значенням якого є номер шуканого відправлення. У відповідь отримаєте json об'єкт із полем *status*, у разі *false* означатиме, що даного номера немає в базі. В протилежному випадку дані будуть представлені, як в прикладі нижче:

```
{
  "current_city": "New York",
  "date_arrive": "Sunday, 03.06.2018 21:54:12",
  "date_departure": "Friday, 01.06.2018 15:23:34",
  "num": 568903,
  "price": 234.46,
  "receiver": "John Doe",
  "sender": "Uncle Tom",
  "status": true
}
```

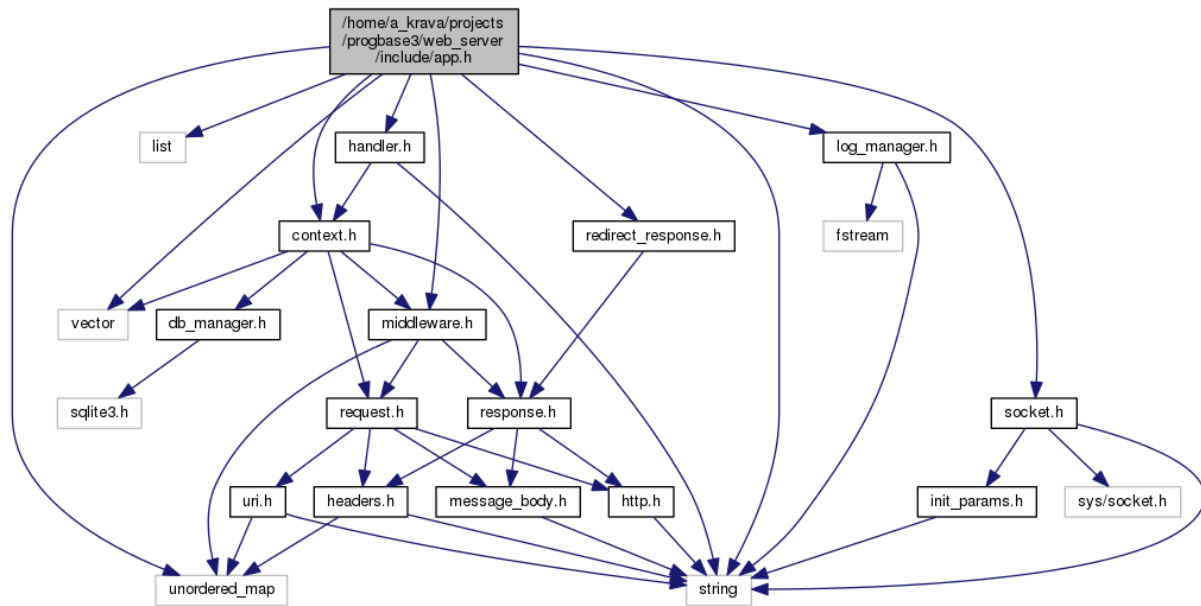


## Опис модулів програми

Фреймворк розбитий на такі модулі:

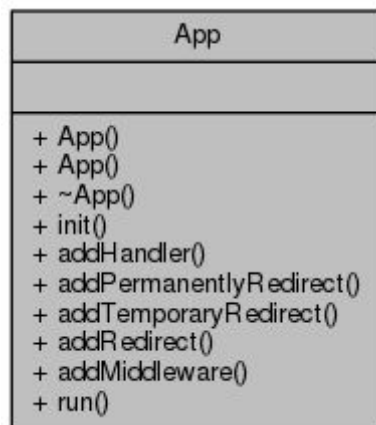


Діаграма залежностей:



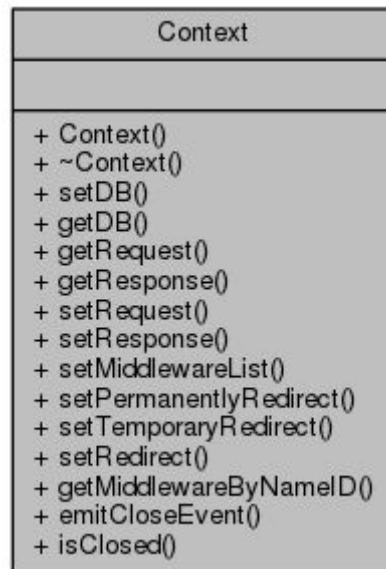
Усі модулі детально описані в документації до фреймворку, яка додана в Додатки.

### Модуль App



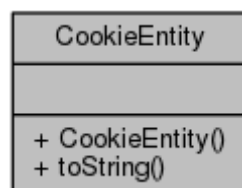
Основний клас системи. Кожен об'єкт цього класу - це незалежна веб-програма, яка може бути налаштована обробниками, проміжним програмним забезпеченням тощо. Цей клас реалізує веб-додаток, що працює на даному і порт. Він підтримує IPv6 і може фіксувати журнал у файлі, якщо вказано. Використовуйте Handlers, Middleware та встановіть переспрямування, щоб налаштувати його.

## Модуль Context



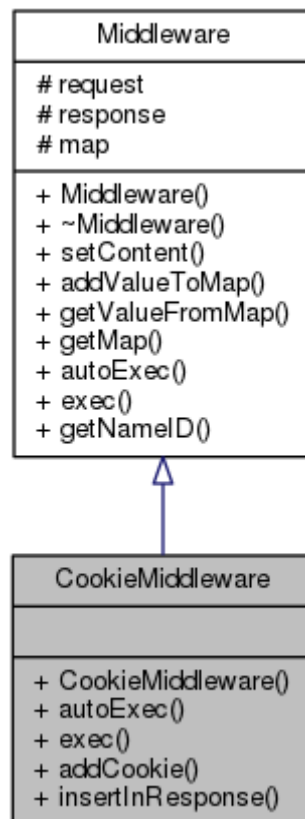
Цей клас є обгорткою для важливих даних (наприклад, Response, DB, Middleware тощо), яка необхідна для обробників. Цей клас збирає дані про поточний запит, який було проаналізовано з str, мати вказівник на об'єкт Response, який в майбутньому буде серіалізований для клієнта (тут можуть бути деякі письмові дані з попередніх відповідей), також є посилання на все додане проміжне програмне забезпечення (ви можете отримати деякі за ідентифікатором) та бази даних, яка готова виконувати метод ехес.

## Модуль CookieEntity



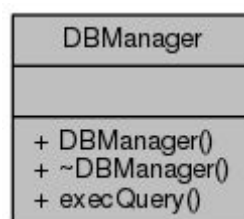
Обкладинка класу для файлів "Cookies". Дозвольте вам налаштувати параметри кожного http-файлу cookie. Використовується CookieMiddleware. Об'єкт цього класу складається з пари ключових значень, а деякі параметри для нього, наприклад, закінчується дата, максимальний вік, домен, шлях, опція http

## ***Модуль CookieMiddleware***



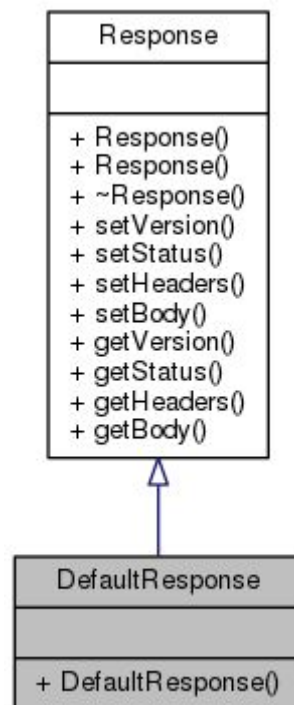
Успадкований клас для аналізу файлів cookie із запиту http. `CookieMiddleware` призначений для аналізу файлів cookie з http-запиту, заповнення відповідей файлами cookie.

## ***Модуль DBManager***



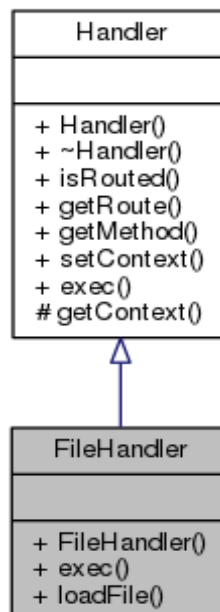
Дозволяє виконувати sql-запити до db. Реалізація обгортки для бази даних SQLite

## Модуль *DefaultResponse*



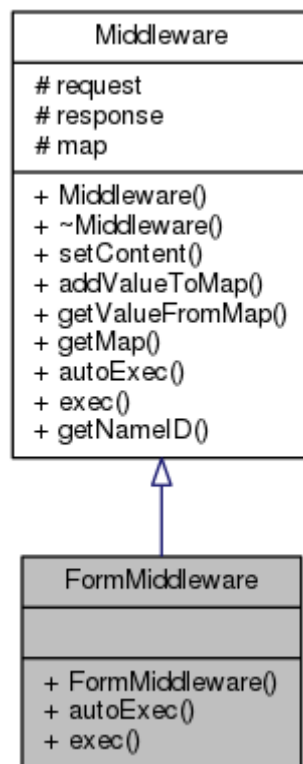
Клас відповіді, який призначений для створення зразків HTML-сторінок на кодах стану. Успадкований клас `DefaultResponse` від `Response` для встановлення заглушок для нерозуміння функціональності.

## Модуль *FileHandler*



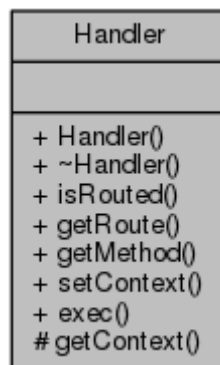
Цей клас дозволяє встановити будь-який файл файлової системи як тіла відповіді. `FileHandler` може працювати як текстові файли (наприклад, `css`, `js`), як двійкові дані (`img`, `png` інші)

## Модуль *FormMiddleware*



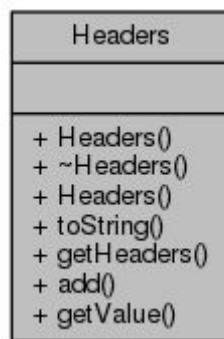
Успадкований клас для аналізу x-www-form-urlencoded. `FormMiddleware` призначений для аналізу форм із запиту http і декодування

## Модуль *Handler*



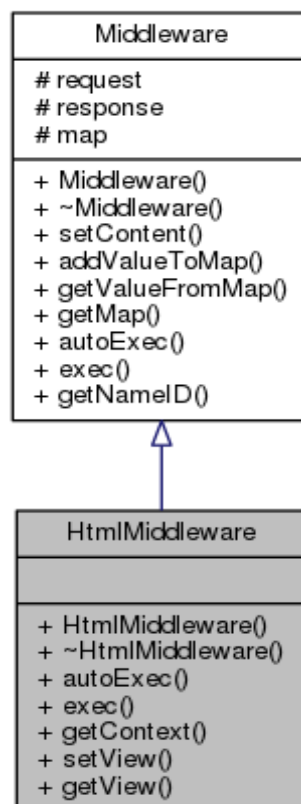
Об'єкт цього класу виконується кожного разу на новий запит, цей об'єкт (та інші) створює відповідь клієнту. Об'єкт обробника може бути загальним (буде виконуватись у кожній відповіді) або налаштований на певний шлях url. Він може отримати всю інформацію про запит, використовувати додаткове проміжне програмне забезпечення та зробити відповідь.

## Модуль *Headers*



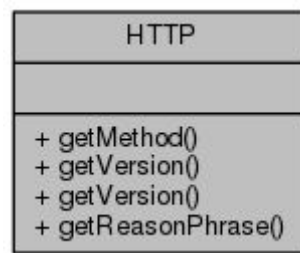
Клас оболонки для заголовків http. Заголовки складаються з карти з пар ключ-значення, і використовується для HTTP-об'єктів запиту та відповіді.

## Модуль *HtmlMiddleware*



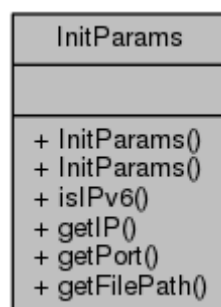
Успадкований клас для відтворення HTML-сторінок із шаблонів. `HtmlMiddleware` використовує логічні шаблони для вусиків для відтворення HTML-сторінок.

## Модуль HTTP



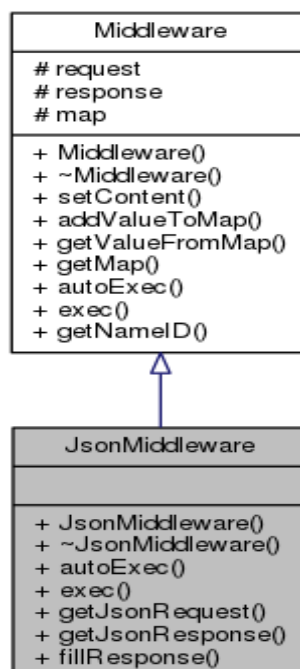
Статичний клас описує http-метод, версію і дозволяє перетворювати його. Клас HTTP описує Method, Version, ReasonPhrase коду в http

## Модуль InitParams



InitParams призначено для отримання конфігурацій веб-сервера з аргументів командного рядка. Цей клас робить перевірку ip-адреси, порту тощо.

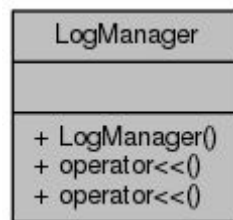
## Модуль JsonMiddleware





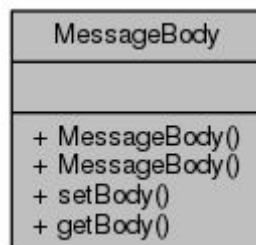
Успадкований клас для виконання будь-яких дій із даними json. JsonMiddleware призначений для аналізу json з http-запиту, заповнення відповідей з json та виконання будь-яких дій із json.

### Модуль *LogManager*



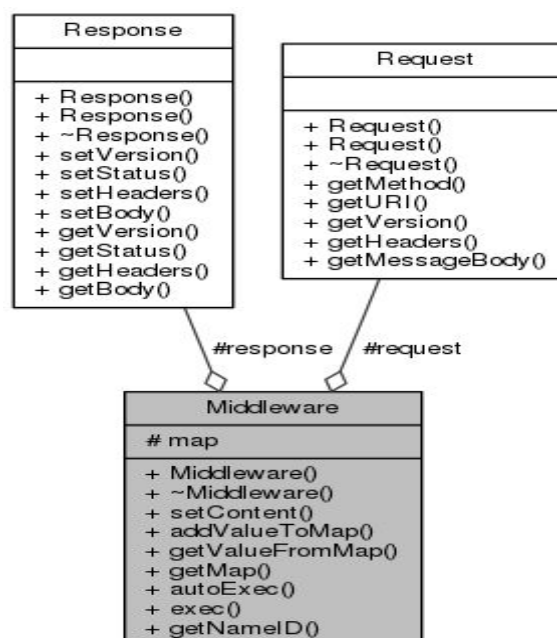
LogManager створює файл і додає в нього логуючу інформацію.

### Модуль *MessageBody*



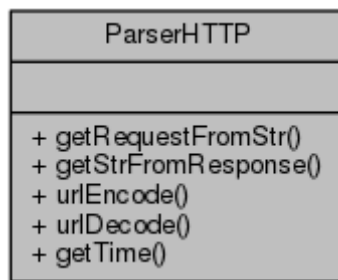
Обгортковий клас для http body. MessageBody містить декодовану інформацію про body http.

### Модуль *Middleware*



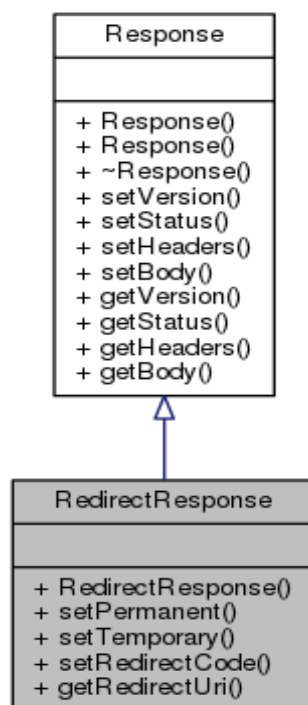
Обгортка класу для проміжного програмного забезпечення. Middleware отримали поточні об'єкти запитів і відповідей, а також карту для пар ключ-значення. Метод `exes` та `autoExes` можуть використовувати об'єкти запиту та відповіді для виконання дій.

### ***Модуль `ParserHTTP`***



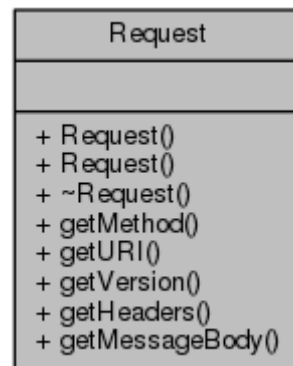
Статичний клас для аналізу, кодування, декодування будь-яких http-даних. `ParserHTTP` використовується для серіалізації та десеріалізації HTTP запитів, відповідей тощо.

### ***Модуль `RedirectResponse`***



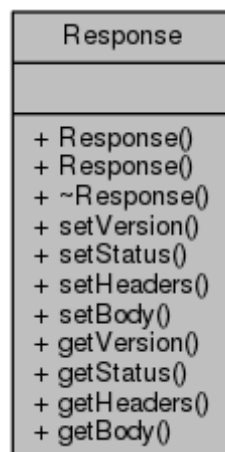
Клас відповіді, який призначений для перенаправлення http. Успадкований клас `RedirectResponse` від `Response` для найлегшого коригування перенаправлень.

## ***Модуль Request***



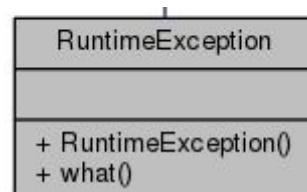
Обгортка класу HTTP-запиту. Об'єктом цього класу є десеріалізований http-запит, де всі дані, що складаються, представлені об'єктами іншого класу.

## ***Модуль Response***



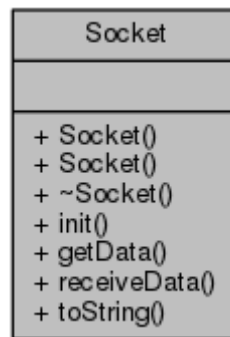
Обгортка класів HTTP-відповіді. Об'єктом цього класу є представлення http-відповіді.

## ***Модуль RuntimeException***



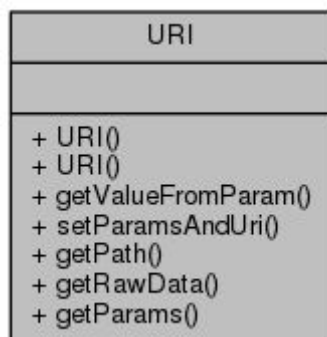
Клас винятку для помилок програми. Спадковий клас із std :: exception

## ***Модуль Socket***



Обгортка функції для передачі / отримання даних через веб-сокети.  
Впровадження функцій сокетів для ОС Linux.


## ***Модуль URI***



Клас представляє http uri. URI складається з uri - рядка uri без аргументів, а також карта пар ключ-значення, що є десеріалізованими параметрами uri.

## Структура програми

Розроблений демо-сервер:



Logistic company

Журнал квитанцій:  
568903 672496 759432 284235

Про компанію Новини Правова інформація Feedback API

Відстежити вантаж

Номер квитанції

Вартість доставки

Терміни доставки

Найближче відділення

Графік роботи кол-центру

Виклик кур'єра

Залиште feedback

Ім'я


Email

Відгук

Відправити відгук

Очистити

Web-framework (c++) 2018



Logistic company

Журнал квитанцій:  
568903 672496 759432 284235

Про компанію Новини Правова інформація Feedback API

Відстежити вантаж

Номер квитанції

Вартість доставки

Терміни доставки

Найближче відділення

Графік роботи кол-центру

Виклик кур'єра

Виклик кур'єра з сайту

Місто 

Beijing

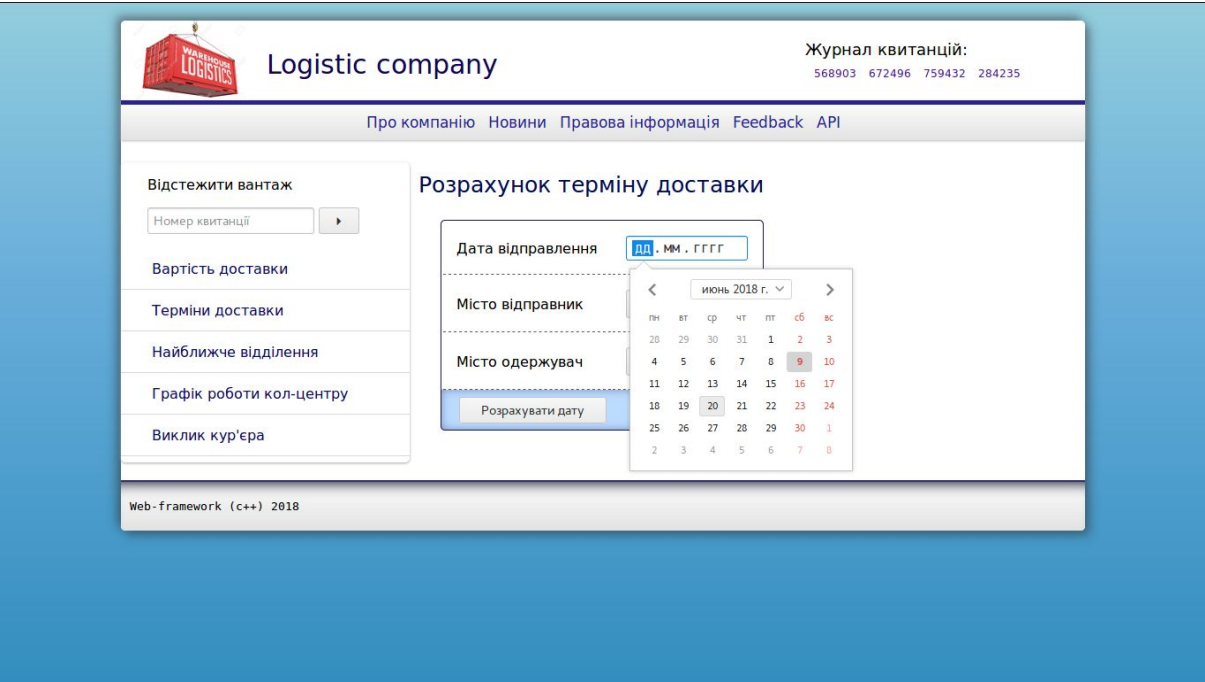
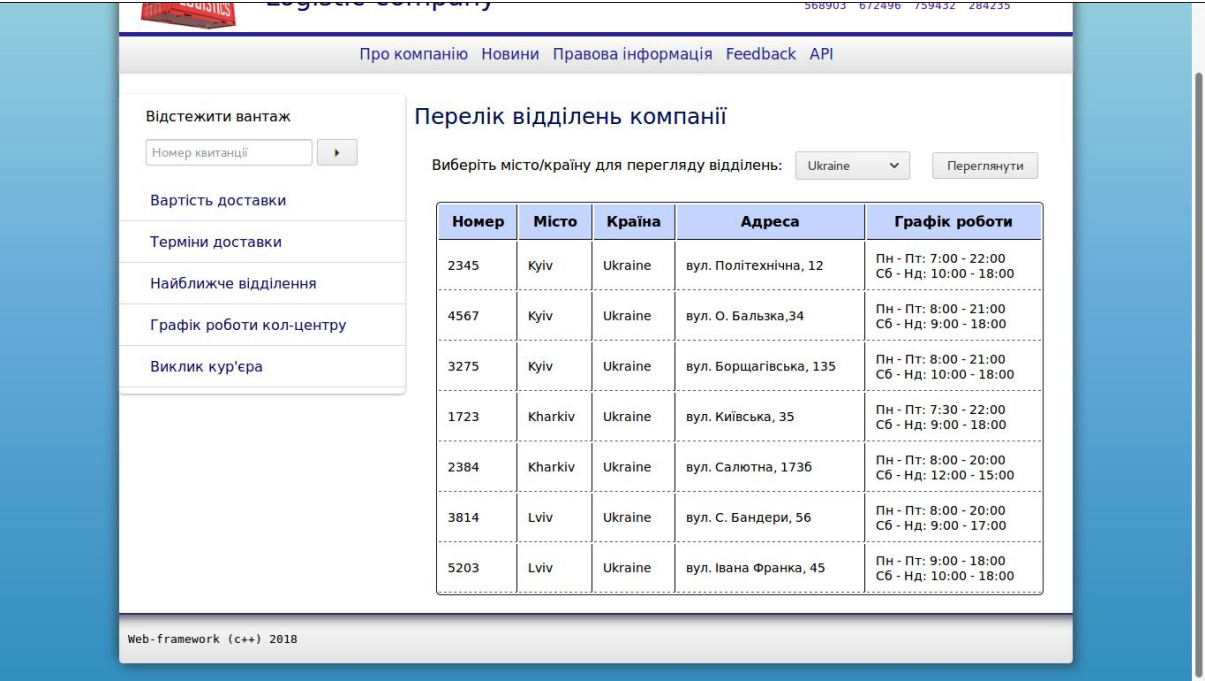
П. І. Б.

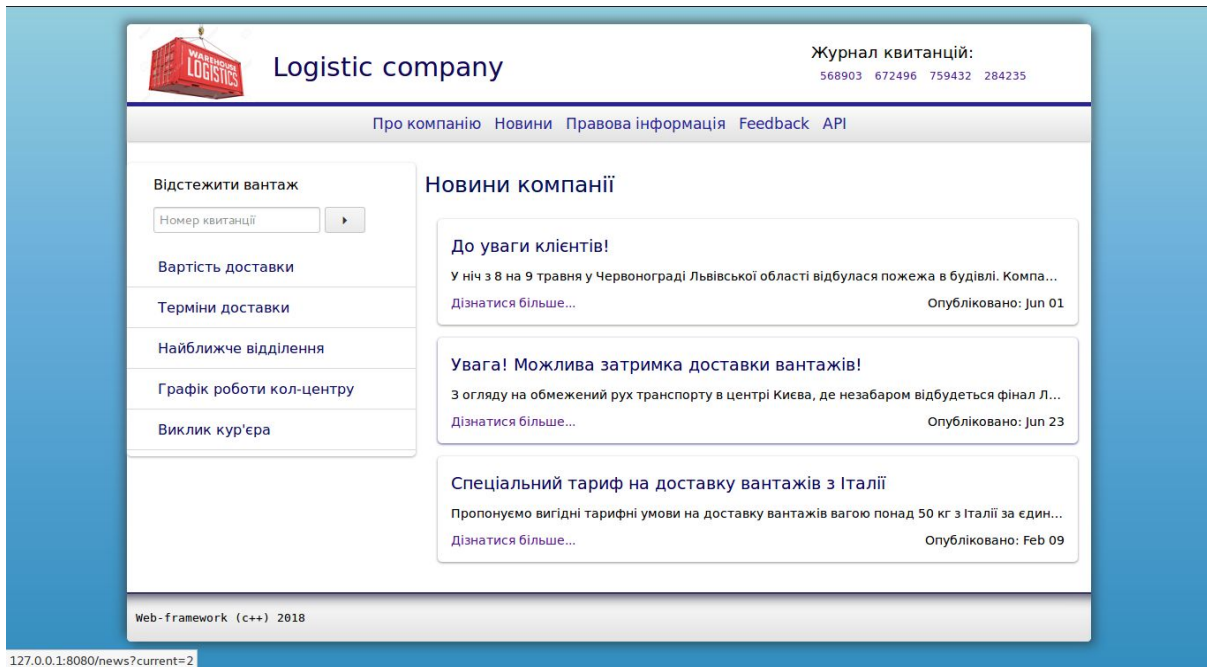
Email

Телефон  формат: +38(0xx)xxx-xx-xx

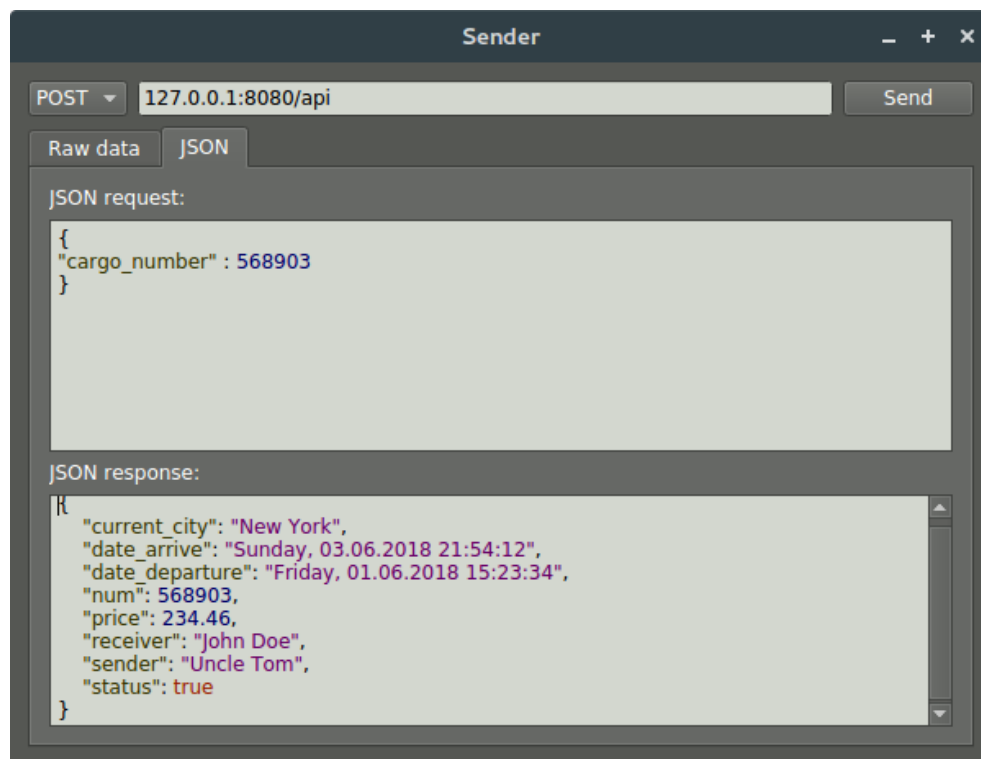
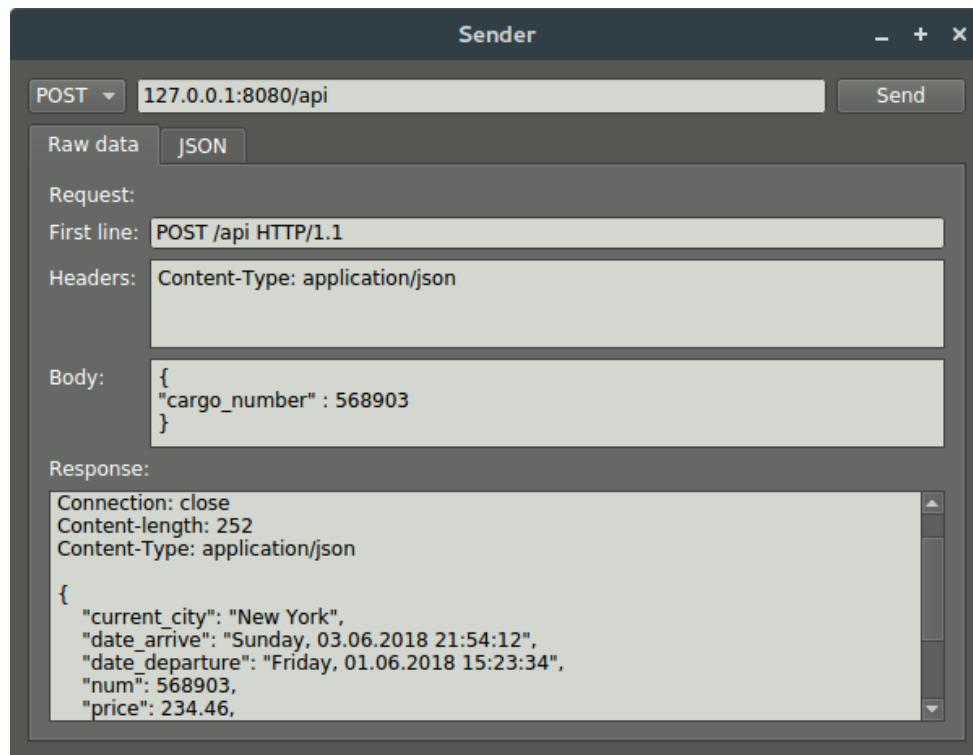
Адреса

Коментар





## Графічний Клієнт:





## Висновки

У результаті було створено веб-фреймворк, яким можна користуватися для того, щоб розробляти власні веб-сервери. Доказом цього став розроблений на основі фреймворку демонстраційний веб-портал для логістичної компанії, який здатен оброблювати будь-які вхідні дані: json, запити від форм тощо. Завдяки підключенню бази даних редагування та внесення контенту до сайту стало набагато простішим. А отже, розроблений комплекс програм є дійсно актуальним та затребуваним сьогодні. Слід відзначити, що завдяки підтримці даних формату json, веб-сервер дійсно може стати універсальним: оброблювати запити та відправляти дані не тільки клієнтам веб-браузерів, а й іншим прикладним програмам, що мають вихід до Інтернету. Розроблений графічний клієнт надає можливість самостійного формування http запитів до веб-серверу з атомарних частин: лінії прешого рядка, заголовочних полів, тіла запиту. Причому відповідь також можна побачити в “raw” вигляді зі всіма її складовими. А можливість редагування та відправки json демонструє прикладну програму, що звертається до веб-серверу за необхідною інформацією. Отже, розроблений програмний комплекс демонструє потужність та універсальність веб-серверів, які створюються на базі розробленого веб-фреймворку.

Написання даної курсової роботи поліпшило мої знання та навички у програмуванні. Було набуто безцінний досвід у створенні власного програмного забезпечення. Знання, необхідні для створення правильної та зрозумілої для інших документації, засвоїлися на практиці завдяки використанню Doxygen. Також поліпшились навички у створенні серверної архітектури програм, було набуто багато нових знань про протокол HTTP

та мережеві технології. Отже, розробка курсової роботи закріпила набуті знання з теорії курсу “Основи програмування”.