ЛАБОРАТОРНАЯ РАБОТА №6

**По курсу «Интернет-программирование»**

Архитектура REST

**Цель работы.**

Ознакомиться с архитектурой REST.

**Приобретаемые компетенции:**

1. Понимание принципов создания интернет-приложений c использованием архитектуры REST.
2. Получение практического опыта проектирования REST API.

# **Принципы технологии REST**

**REST** - (REpresentational State Transfer) — это архитектура, т.е. принципы построения распределенных гипермедиа систем, того что другими словами называется World Wide Web, включая универсальные способы обработки и передачи состояний ресурсов по HTTP.

**Архитектура REST позволяет добиться в приложении:**

* Масштабируемости
* Общности интерфейсов
* Независимости внедрение компонентов
* Применения промежуточных компонентов, снижающих задержку, усиливающие безопасность

**Можно рекомендовать использовать REST в случаях:**

* Ограничений пропускной способности соединения. (Экономия трафика)
* Необходимости кэшировать запросы
* Предвидится значительное масштабирование системы
* В сервисах, использующих AJAX

**Преимущества REST:**

* Отсутствие дополнительных внутренних прослоек, что означает передачу данных в том же виде, что и сами данные. Т.е. данные не оборачиваются в XML, как это делает SOAP и XML-RPC, не используется AMF, как это делает Flash и т.д. Данные отдаются с минимальным количеством символов форматирования.
* Каждая единица информации (ресурс) однозначно определяется URL — это значит, что URL по сути является первичным ключом для единицы данных. Причем совершенно не имеет значения, в каком формате находятся данные по адресу — это может быть и HTML, и jpeg, и документ Microsoft Word.
* Как происходит управление информацией ресурса — это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол - HTTP. Для HTTP действие над данными задается с помощью методов : GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить).

Таким образом, действия (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST. При этом существует возможность выполнять все запросы с параметрами. Поэтому даже с помощью браузеров, поддерживающих только методы GET и POST протокола HTTP возможна создание полноценного API, включающего методы добавления, удаления и изменения данных.

# Пример проектирования архитектуры REST

**Пусть есть список писем в БД, представленной в формате JSON-файла:**

{

"mail1": {

"datatime":"07.05.2020 10:10:15",

"subject":"Первое",

"from":"andymailru@mail.ru",

"message":"Первая проверка",

"files" :{}

},

"mail2": {

"datatime":"07.05.2020 10:15:15",

"subject":"Второе",

"from":"andymailru@mail.ru",

"message":"Вторая проверка",

"files" :{}

},

"mail3": {

"datatime":"07.05.2020 10:20:15",

"subject":"Третье",

"from":"andymailru@mail.ru",

"message":"Третья проверка",

"files" :{}

}

}

Необходимо спроектировать REST API, позволяющий: добавлять и получать письма из этой структуры.

Первый этап проектирование REST API можно осуществлять без привязки к конкретному языку программирования на котором, впоследствии, будет написано API. Опишем методы получения списка почтовых сообщений (GET) и добавления почтового сообщения (PUT).

## Таблица 1. Методы REST API

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задача | URI | Метод | Тело запроса | Успех | Возможные ошибки, отличные от стандартных (см. Коды ошибок) и их коды |
| Получить список всех писем | /getMailList | GET |  | JSON с последними 50-ю письмами |  |
| Письма с subject, включающим *string* |  |  | action=filter&filter={"subject":"*string*"} | JSON с отфильтрованными письмами |  |
| Добавление писем |  |  | action=put&data={ "datatime":"*d.m.Y H:i:s*", "subject":"string", "from":"email", "message":"string" } | Код добавленного письма, например, mail\_4 |  |
| Добавить одно письмо | /addMail | PUT | { "datatime":"d.m.Y H:i:s", "subject":"string", "from":"email", "message":"string" } | Код добавленного письма, например, mail\_4 | 500 - Ошибка валидации входных данных |
|  |  |  |  |  |  |

## Коды некоторых распространенных ошибок

403 — Сервис не доступен

404 — Данные недоступны

500 — Не идентифицированная ошибка

Полный список ошибок - <https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP>

## Код примера на JavaScript (под Node JS) реализующий методы Таблицы 1.

Для реализации REST API, можно использовать модуль express, который обеспечиdает создание http сервера и обработку запросов методов: GET, POST, PUT, DELETE с помощью методов вида:

app.***метод***('***URI***l', function(req, res) {

//Обработка запроса — req (REQUEST)

//Воврат результата через res (RESPONSE)

});

где:

app – экземпляр объекта, созданного сервера

***метод*** — может принимать значения: **get, post, put, delete**

***URI*** — идентификатор ресурса из таблицы, например **/addMail**

Код примера на JavaScript приведен в Приложении 1.

Входные JSON данные проверяются с помощью модуля ajv (см. описание модуля в табл. 2.) на основании схемы из файла mailShema.json ( см. Приложение 2).

БД представляет собой JSON файл с именем maillist.json ( см. Приложение 3). Для работы с БД используется модуль fs.

Обработка ошибок реализована с помощью модуля http-errors, возвращаемых через метод приложения sendErr.

Все модули устанавливаются из командной строки с помощью пакетного менеджера npm командой вида: **npm install *имя\_модуля***.

Для проверки работы используется браузер (для метода GET) и консольная программа curl для методов GET и PUT. Пример команды PUT приведен в Приложении 1 в комментариях к методу app.put(…). Документация по curl: <https://curl.haxx.se/docs/manpage.html> (англ.), <http://osxh.ru/terminal/command/curl> (рус), <https://tokmakov.msk.ru/blog/item/191> (примеры запросов).

## Таблица 2. Документация по модулям Node.js

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Источник** |
| express | Фреймворк http сервера | <https://metanit.com/web/nodejs/4.1.php> |
| fs | Работа с файлами на диске | <https://metanit.com/web/nodejs/2.8.php> |
| http-errors | Работа с ошибками http | <https://github.com/jshttp/http-errors> |
| ajv | Валидация JSON данных на основе JSON-схем | <https://www.npmjs.com/package/ajv> |

**Задание на лабораторную работу**

1. Изучить код примера в Приложении 1
2. Выполнить (по собственному выбору) один из вариантов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Задание | Оценка | | |
| 1 | Реализовать пример из Приложения 1 на другом языке: PHP, Java, Python |  | 4 |  |
| 2 | Добавить в Пример 1 удаление письма по коду (метод DELETE) | 3 | 4 | 5 |
| 3 | Добавить в Пример 1 добавление нескольких писем, как методом PUT так и методом POST (чтобы работало через Браузер) |  |
| 4 | Добавить в Пример 1 передачу файлов в аттачмете (поле files) письма |  |  |

**Контрольные вопросы:**

1. Что означает REST?
2. Почему REST называют архитектурой?
3. Когда рекомендуется используется REST?
4. Каковы преимущества REST?
5. Каковы недостатки REST?
6. В чем отличие методов GET и POST?
7. В чем отличие методов GET и PUT?

# Приложение 1

# Код примера на языке JavaScript под управлением Node JS, реализующей REST API

/\* Загружаем модули, создаем объекты \*/

var express = require('express'); //модуль, для создания http-сервера

var fs = require("fs"); //модуль для работы с файлами

const createError = require('http-errors'); //модуль для формирования сообщений об ошибках

const jsonParser = express.json(); // создаем парсер для данных в формате json

const ajv = require('ajv'); //модуль для проверки JSON данных на основе JSON схемы

var eajv = new ajv();

var debug = false; //Режим отладки, когда отправляется стек устанавливаем в false

//Запускаем сервер по адресу http://localhost:8081/

var app = express();

var server = app.listen(8081, function () {

var host = "localhost";

server.address().address = host;

var port = server.address().port;

console.log("Example app listening at http://%s:%s", host, port)

})

//Обработка GET/getMailList

app.get('/getMailList', function (req, res) {

let Result = "";

console.log(req.query);

let action = req.query.action;

switch(action) {

case 'filter':

console.log(req.query.filter);

Result = makeFilter(req.query.filter);

break;

default:

Result = getAllDate();

break;

}

res.end(makeHTML(Result));

})

//Обработка GET запроса

app.get('/addMail', function(req, res) {

//Получаем JSON с данными письма

let Result = "<hr>"+JSON.stringify(req.query)+"<hr>"+req.body

console.log(req.query);

console.log(req.body);

res.end(makeHTML("<b>GET Success</b>"+Result));

});

//Обработка PUT/addMail запроса

app.put('/addMail', jsonParser, function(req, res) {

/\*

Образец запроса

curl --header "Content-Type: application/json" --request PUT

--data {\"datatime\":\"d.m.Y&nbsp;H:i:s\",\"subject\":\"string\",\"from\":\"email\",\"message\":\"string\"}

http://localhost:8081/addMail?action=add

Сокращенный запрос

curl -H "Content-Type: application/json" PUT

-d {\"datatime\":\"d.m.Y&nbsp;H:i:s\",\"subject\":\"string\",\"from\":\"email\",\"message\":\"string\"}

http://localhost:8081/addMail?action=add

\*/

try {

increment = addNewMailToDB(req.body);

let Result = "id:mail\_"+increment+"<hr>"+JSON.stringify(req.query)+"<hr>"+JSON.stringify(req.body);

console.log(req.query);

console.log(req.body);

res.end(makeHTML("<b>PUT Success</b>"+Result));

} catch(e) {

sendErr(e);

}

});

//Метод добавляет новое письмо в БД

function addNewMailToDB(mail) {

try {

//Проверяем входные данные

var valid = eajv.validate(JSON.parse(fs.readFileSync(\_\_dirname+'/mailShema.json', "utf8")), mail);

if (!valid) { throw new createError(500, `Error validation input data`); }

//Добавляем запись в БД

let mailList = JSON.parse(getAllDate());

let increment = mailList.header.increment\*1+1;

mailList.body["mail\_"+increment] = mail;

mailList.header.increment = increment;

saveAllDate(JSON.stringify(mailList, null, 4));

return increment;

} catch(e) {

throw e;

}

}

//Метод создает HTML с ответом

function makeHTML(data) {

let Result = '<html><head><meta charset="utf-8"></head><body>'+data+'</body></html>';

return Result;

}

//Выполнение фильтрации по subject

function makeFilter(param) {

let Result = "makeFilter";

let filter = JSON.parse(param);

let data = JSON.parse(getAllDate());

data = data.body;

//Поиск по названию

if( filter.subject != undefined ) {

for (key in data) {

console.log(data[key].subject+" "+filter.subject);

let temp = data[key].subject;

if( temp.indexOf(filter.subject) >= 0 ) {

Result += JSON.stringify(data[key]);

}

}

}

return Result;

}

//Метод загружает данные из БД

function getAllDate() {

let Result;

try {

Result = fs.readFileSync(\_\_dirname+'/maillist.json', "utf8");

return Result;

} catch (e) {

sendErr(e, `Error BD access`);

}

}

//Метод сохраняет данные в БД

function saveAllDate(data) {

fs.writeFile(\_\_dirname+'/maillist.json', data, function(error){

if(error) sendErr(e, `Error BD access`); // если возникла ошибка

});

return true;

}

//Метод канализирует обработку и отправку ошибок

function sendErr(e, msg, code) {

if( !debug && e.stack != undefined ) e.stack = ''; //если не режим отладки - очищаем информацию о стеке

if( msg != undefined ) e.message = msg;

if( code != undefined ) e.code = code;

throw createError(e.code, e);

}

# Приложение 2. mailShema.json – файл, содержащий JSON схему для проверки входных данных

{

"description": "A schema for mail",

"type":"object",

"properties":{

"datatime":{

"description":"Дата и время",

"type":"string"

},

"subject":{

"description":"Подпись по оси X",

"type":"string"

},

"from":{

"description":"Подпись по оси Y",

"type":"string"

},

"message":{

"description":"Тип данных по оси X",

"type":"string"

}

},

"required": ["datatime", "subject", "from", "message"]

}

# Приложение 3. maillist.json - **ф**айл БД примера из Приложения 1

{

"header": {

"date\_create": "07.05.2020",

"date\_update": "07.05.2020",

"author": "aam",

"increment": 3

},

"body": {

"mail\_1": {

"datatime": "07.05.2020 10:15:15",

"subject": "Первое",

"from": "andymailru@mail.ru",

"message": "Первая проверка"

},

"mail\_2": {

"datatime": "07.05.2020 10:15:15",

"subject": "Второе",

"from": "andymailru@mail.ru",

"message": "Вторая проверка",

"files": {}

},

"mail\_3": {

"datatime": "07.05.2020 10:20:15",

"subject": "Третье",

"from": "andymailru@mail.ru",

"message": "Третья проверка",

"files": {}

}

}

}