Урок N6
REST API

REST API определяет набор функций, к которым разработчики могут совершать запросы и получать ответы, как правило по протоколу HTTP(S). По сути REST API принципиально отличается от обычного веб-приложения лишь тем, что проектируется для предоставления программным клиентам данных в "чистом виде", без их оформления с помощью HTML и CSS, в удобном для обработки формате.

Содержание урока

CRUD

В концепции REST принципиальное важно то, что каждый запрос на сервер содержит все необходимые данные для предоставления сервером полного ответа. Проще говоря, один запрос на сервер = один результат. Самый популярный пример использования REST - протокол HTTP. Также в традиционных REST API как правило не используются куки и сессии как способ передачи "состояния" на сервер - для этого достаточно данных в самом запросе.

При ответе на HTTP запросы сервер получает информацию не только о заголовках запроса и его содержимом, но и о запрашиваемом адресе и о выбранном методе запроса. Для создания REST API полезно знать, какие HTTP методы с какими действиями обычно ассоциируются.

Create - POST
Read - GET
Update - PUT
Delete - DELETE

Запросы PUT и DELETE работают так же как и POST-запросы, но отличаются по смыслу выполняемой операции. Например, при GET запросе к /objects/1 сервер вернет информацию об объекте. При PUT запросе по тому же адресу он может обновить информацию об объекте на основании полученных данных. При DELETE запросе объект будет удален.

Правильное использование HTTP методов совместно с ЧПУ (человек-понятных УРЛ, "красивых ссылок") позволяет создать интуитивно понятный REST API.

Поскольку REST API это по-прежнему обычное веб-приложение, для использования REST API прекрасно подходит модуль request из предыдущих уроков. Также можно создать REST API с помощью стандартных функций работы с сетью или модуля express. В этом уроке мы рассмотрим похожий на express модуль restify, который предназначен специально для использования и создания REST API.

Пример подключения к API Google Maps с помощью restify (клиент):

```
var util = require('util');
var restify = require('restify');

var client = restify.createJsonClient({
	url: 'http://maps.googleapis.com/'
});

client.get(
	'/maps/api/geocode/json?address=Moscow+Red+Square',
	function(err, req, res, obj) {
		if (err)
			console.error(err);
		else
			console.log(util.inspect(obj, {depth: null}));

		var components = obj.results[0].address_components;
		for (key in components) {
			if (components[key].types == "postal_code") {
				console.log('ZIP CODE: ');
				return console.log(components[key].short_name);
			}
		}
	}
);
```

Идейно restify очень похож на express.
В нем точно так же используются middleware и роутинг:

```
var util = require('util');
var restify = require('restify');

rest = restify.createServer({
	name: 'ProgSchool'
});

rest.use(function(req, res, next) {
	if (req.path().indexOf('/protected') != -1) {
		var auth = req.authorization;
		if (auth.scheme == null) {
			res.header('WWW-Authenticate', 'Basic realm="Hi!"');
			return res.send(401);
		}
	}

	return next();
}

rest.use(restify.authorizationParser());
rest.use(restify.queryParser());
rest.use(restify.bodyParser());
rest.use(restify.gzipResponse());

rest.get('/', function(req, res) {
	res.send(200, {result: "OK"});
});

rest.get('/protected/export/:format', function(req, res) {
	res.send(200, {result: req.params.format});
});

rest.listen(8080, function() {
	console.log('API launched');
});
```

CORS

REST API часто получают и предоставляют данные в "родном" формате JSON и их удобно использовать в т.ч. из JS-скриптов, которые выполняются на стороне клиента в браузере и взаимодействуют с сервером через AJAX.

Из соображений безопасности по умолчанию браузеры запрещают отправлять AJAX-запросы, если исходный и целевой домены не совпадают. Чтобы иметь возможность реализовывать кроссдоменные запросы, современные браузеры поддерживают стандарт CORS.

При попытке совершения кроссдоменного запроса из клиентского JS-кода, браузер сначала отправит на сервер т.н. preflight-запрос с помощью метода OPTIONS чтобы узнать у сервера, имеет ли он право посылать к нему запрос с другого домена. Если сервер ответит с помощью специальным образом сформированных заголовков, браузер следующим запросом отправит исходный запрос на сервер.

Пример реализации REST API на restify с поддержкой CORS вы найдете в исходных кодах к уроку в файле producer.js

Самоконтроль

✓ CRUD и базовые методы HTTP
✓ restify для использования сторонних REST API
✓ restify для создания собственных REST API
✓ AJAX-запросы из браузера и CORS

Домашнее задание

1) С помощью созданной в предыдущем уроке модели работы с объектами на сервере создайте REST API для просмотра данных с помощью метода GET.

2) Добавьте в приложение возможность импорта данных с помощью POST, модификации данных с помощью PUT и их удаления с помощью DELETE.