Объявление маршрутов в модуле

Первый шаг организации маршрутов — размещение их в собственном модуле. Существует несколько способов сделать это. Один из способов — создать для вашего модуля функцию, возвращающую массив объектов, содержащих свойства `method` и `handler`. Затем вы можете описать маршруты в файле своего приложения следующим образом:

```
var routes = require('./routes.js')();

routes.forEach(function(route){
    app[route.method](route.handler);
})
```

У этого способа есть свои достоинства, и он может быть приспособлен для динамического хранения маршрутов, например, в базе данных или файле JSON. Однако, если такая функциональность вам не нужна, советую передавать в модуль экземпляр `app` и поручить ему добавление маршрутов. Именно такой подход мы будем использовать в нашем примере. Создадим файл `routes.js` и переместим в него все наши существующие маршруты:

```
module.exports = function(app){
    app.get('/', function(req,res){
        app.render('home');
    }))
    //...
};
```

Иными словами, чтобы передать что-то в модуль, мы а) оборачиваем всё его содержимое функцией с параметрами (или **{options}**) (создавая таким образом пространство имён) б) делаем **require** и дальше оператор вызова функции, в котором передаём то, что надо, например **app**

Если мы просто вырежем их и вставим, то столкнемся с определенными проблемами. Например, наш обработчик `/about` использует недоступный в данном контексте объект `fortune`. Мы можем добавить требуемые импорты, но воздержимся от этого: мы собираемся в ближайшем будущем переместить обработчики в их собственный модуль, тогда и решим данную проблему.

Так как мы скомпонуем маршруты? Очень легко: в `meadowlark.js`, просто импортируем наши маршруты:

```
require('./routes.js')(app);
```

Логическая группировка обработчиков

Чтобы придерживаться первого руководящего принципа (использование именованных функций для обработчиков маршрутов), нам понадобится место, куда можно поместить эти обработчики. Один довольно-таки экстремальный вариант — отдельный JavaScript-файл для каждого обработчика. Мне сложно представить себе ситуацию, при которой данный подход был бы оправдан. Лучше каким-то образом сгруппировать связанную между собой функциональность. Это не только упрощает применение совместно используемой функциональности, но и облегчает внесение изменений в связанные методы.

Пока что сгруппируем нашу функциональность в следующие отдельные файлы: `handlers/main.js`, в который мы поместим обработчик домашней страницы, обработчик страницы О... и вообще любой обработчик, которому не найдется другого

логического места; handlers/vacations.js, в который попадут относящиеся к отпускным турам обработчики, и т. д.

Рассмотрим handlers/main.js:

```
var fortune = require('../lib/fortune.js');

exports.home = function(req, res){
    res.render('home');
};

exports.about = function(req, res){
    res.render('about', {
        fortune: fortune.getFortune(),
        pageTestScript: '/qa/tests-about.js'
    м} );
};

//...
```

Теперь изменим routes.js, чтобы воспользоваться этим:

```
var main = require('./handlers/main.js');
module.exports = function(app){
    app.get('/', main.home);
    app.get('/about', main.about);
    //...
};
```

Это удовлетворяет всем нашим руководящим принципам. /routes.js **исключительно** понятен. В нем легко понять с первого взгляда, какие маршруты существуют в вашем сайте и где они обрабатываются. Мы также оставили себе массу свободного пространства для роста. Можем сгруппировать связанную функциональность в таком количестве различных файлов, какое нам нужно. и если routes.js когда-нибудь станет слишком громоздким, мы можем снова использовать тот же прием и передать объект app другому модулю, который, в свою очередь, зарегистрирует больше маршрутов (хотя это начинает попахивать переусложнением — убедитесь, что столь запутанный подход в вашем случае действительно обоснован).

Автоматическая визуализация представлений