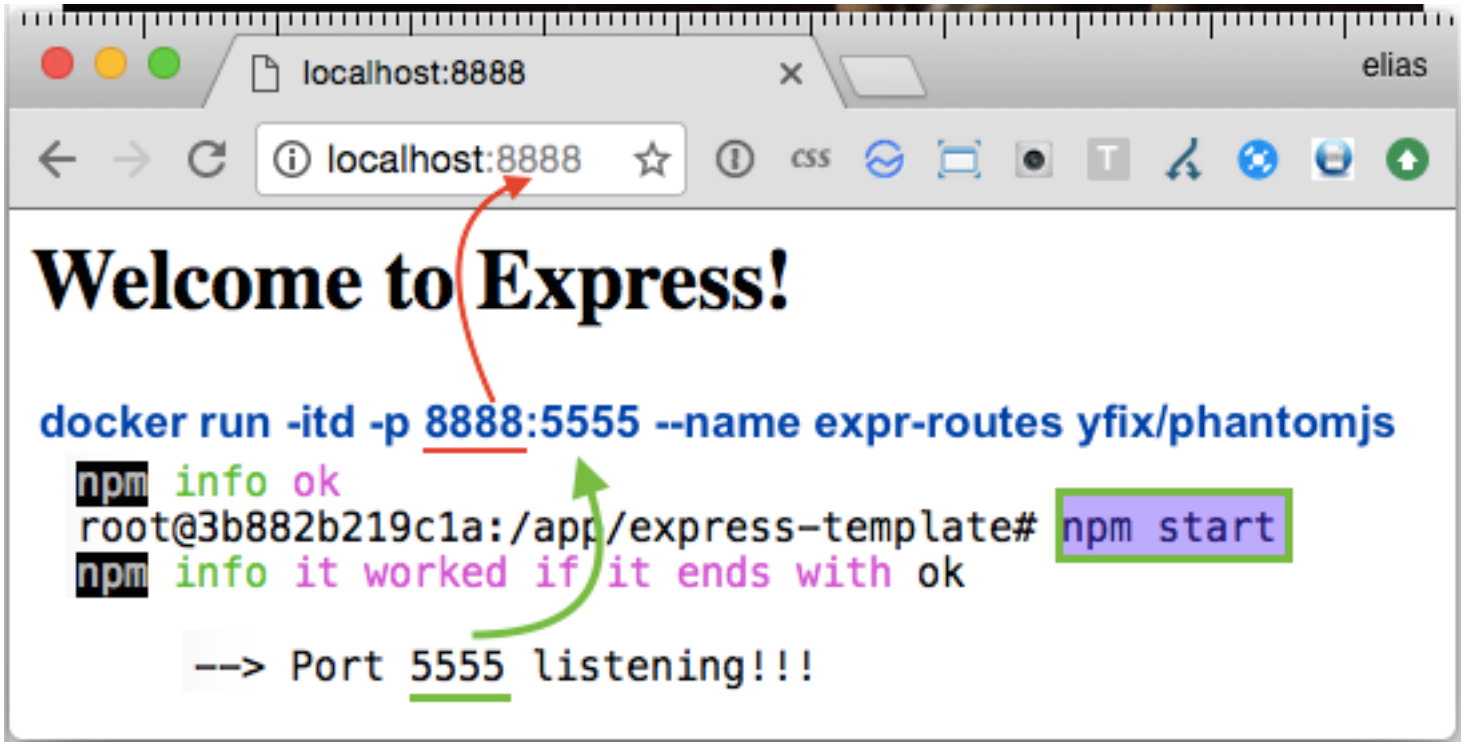
Маршрутизация:

начало

Запуск в Docker

- [docker pull yfix/phantomjs]



Между docker run и npm start

- docker exec -it expr-routes bash
- wget https://raw.githubusercontent.com/gossoudarev/webteach/master/express-template/install.sh
- bash install.sh
- cd express-template
- npm install

Маршрутизация - это

- механизм выдачи клиенту контента по запросу с использованием частей URL
- рассмотрим, как создать набор маршрутов, начинающихся с my

- mkdir routes
- cd routes
- mkdir my
- cd my
- touch index.js

routes/my/index.js

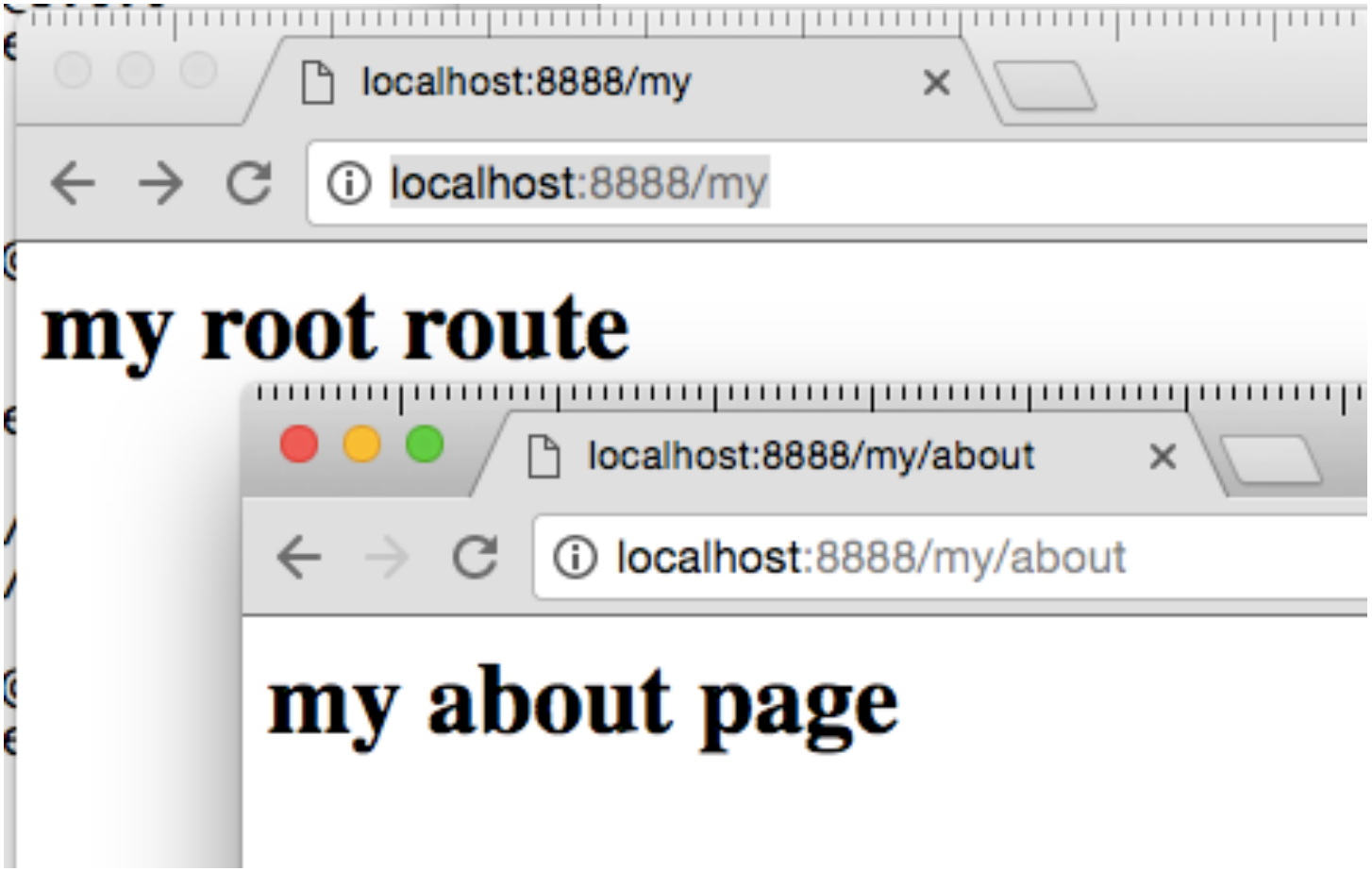
```
1  var router = require('express').Router();
2
3  /* counting from /my/ as root */
4  router.get('/', (req, res ) => {
5              // actually myapp.com/my/
6              res.send('<h1>my root route</h1>');
7  })
8         .get('/about', (req, res) => {
9              res.send('<h1>my about page</h1>');
10 });
11
12 module.exports = router;
```

server.js

- var my_routes = require('./routes/my/');
- …
- app.use('/my', my_routes);
- что мы регистрируем с помощью **use**?
- похоже ли это на ***маунтинг***?

Дерево маршрутов

- Маршрутизатор отвечает за ветку my/*



REST-параметризация

- Как сделать, чтобы по меняющейся части URL выдавать сходные ответы?
- /my/name/Ilya   -> Welcome, Ilya
- /my/name/Pavel -> Welcome, Pavel
- ???

'/name/**:name**'

- req.params.name
- не зависит от глагола
- http://forbeslindesay.github.io/express-route-tester/

Express will compile your route into the following regular expression

```
/^(?:\/(?=$))?$/i
```

Пути маршрутов и регулярные выражения

Когда вы задаете путь (например, `/foo`) в своем маршруте, Express в конечном счете преобразует его в регулярное выражение. В путях маршрутов можно использовать некоторые метасимволы регулярных выражений: `+`, `?`, `*`, `(` и `)`.

Допустим, вы хотели бы, чтобы URL `/user` и `/username` обрабатывались с помощью одного маршрута:

```
app.get('/user(name)?', function(req,res){
    res.render('user');
});
```

Обычно:

GET

req.query.name

POST

```
bodyParser = require('body-parser')
```

req.body.name

- …и приходится подключать body parser и как-то "нормализовать" запросы, приводя к "общему знаменателю"

"нормализация"

```
let normalize = (req, res, next)=>{
  // middleware
  req.what = req.method=='POST' ? req.body : req.query;
  // ...

  next();
};

...

app.all('/....', normalize, (req, res)=>{
    // что-то с req.what

});
```

Когда реализуем REST

- т.е. обращаемся разными глаголами к одному и тому же маршруту (см. далее)
- то сосредотачиваем всё под router.route
- и при использовании параметров мы не зависим от query/body

Общая схема

```
router.route('/:name')
  .all(function (req, res, next) {
    ...
})
   .get(function (req, res) {
     ...
   });
   .delete(function (req, res) {
... });
```

routes/my/index.js


```
var router = require('express').Router();

router.get('/', (req, res) => {
          res.send('<h1>my root route</h1>');
})
      .get('/about', (req, res) => {
          res.send('<h1>my about page</h1>');
})
      .route('/name/:name')
          .get((req, res) => {
             res.send(`<h1>You GET welcome, ${req.params.name}</h1>`);
      })
          .post((req, res) => {
             res.send(`<h1>You POST welcome, ${req.params.name}</h1>`);
      });

module.exports = router;
```

Тестируем

- попробуйте Postman

```
 $ curl -s 'localhost:8888/my/name/Ilya'
<h1>You GET welcome, Ilya</h1>
 $ curl -s --request POST 'localhost:8888/my/name/Pavel'
<h1>You POST welcome, Pavel</h1>
 $ curl -s -X POST 'localhost:8888/my/name/Victor'
<h1>You POST welcome, Victor</h1>
```

- https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop

Порядок имеет значение!

- в начале можно:
  - app.use('/*', all);
  - (функция all – универсальное m/w, не забывайте вызывать next())
- в конце:

```
// 404 catch-all handler (middleware)
app.use(function(req, res, next){
    res.status(404).send('<h1>Not yet here!</h1>');
    // потом: res.status(404).render('404');
})

/* 500 error handler (middleware) */
   .use(function(err, req, res, next){
    console.error(err.stack);
    res.status(500).send('<h1>Sorry! Something went wrong!</h1>');
    // потом: res.status(500).render('500');
});
```

Практика!

- Дополните наше приложение-хэшер (dz2) глаголонезависимой маршрутизацией
- не забывайте о тестах
- используя **request** и **pipe** создайте приложение, выцепляющее главные страницы некоторых сайтов по node и выдающее их по маршрутам /out/npm, /out/express, /out/koa и так далее
- Выполните практическое задание по logger_middleware_codeschool.pdf
- Разберитесь, как обрабатывать запросы к субдоменам (Итан Браун, гл. 14)

Для исследования

- автоматизация машрутов/представлений
- динамизация (считывание из JSON etc)
  - Итон Браун, с.200-202

Два наиболее популярных подхода к организации маршрутов: *маршрутизация с именованной областью видимости* и *ресурсная маршрутизация*. Маршрутизация с именованной областью видимости хорошо зарекомендовала себя в случае наличия большого количества маршрутов, начинающихся с одного и того же префикса (например, `/vacations`). Применение данного подхода облегчает модуль Node `express-namespace`. Ресурсная маршрутизация автоматически добавляет маршруты на основе методов объекта. Это может быть полезной методикой, если логика сайта естественным образом является объектно-ориентированной. Пакет `express-resource` — пример реализации этого стиля организации маршрутов.