Tecnologie e applicazioni web

Express.js

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Academic year: 2023/2024

Express.js

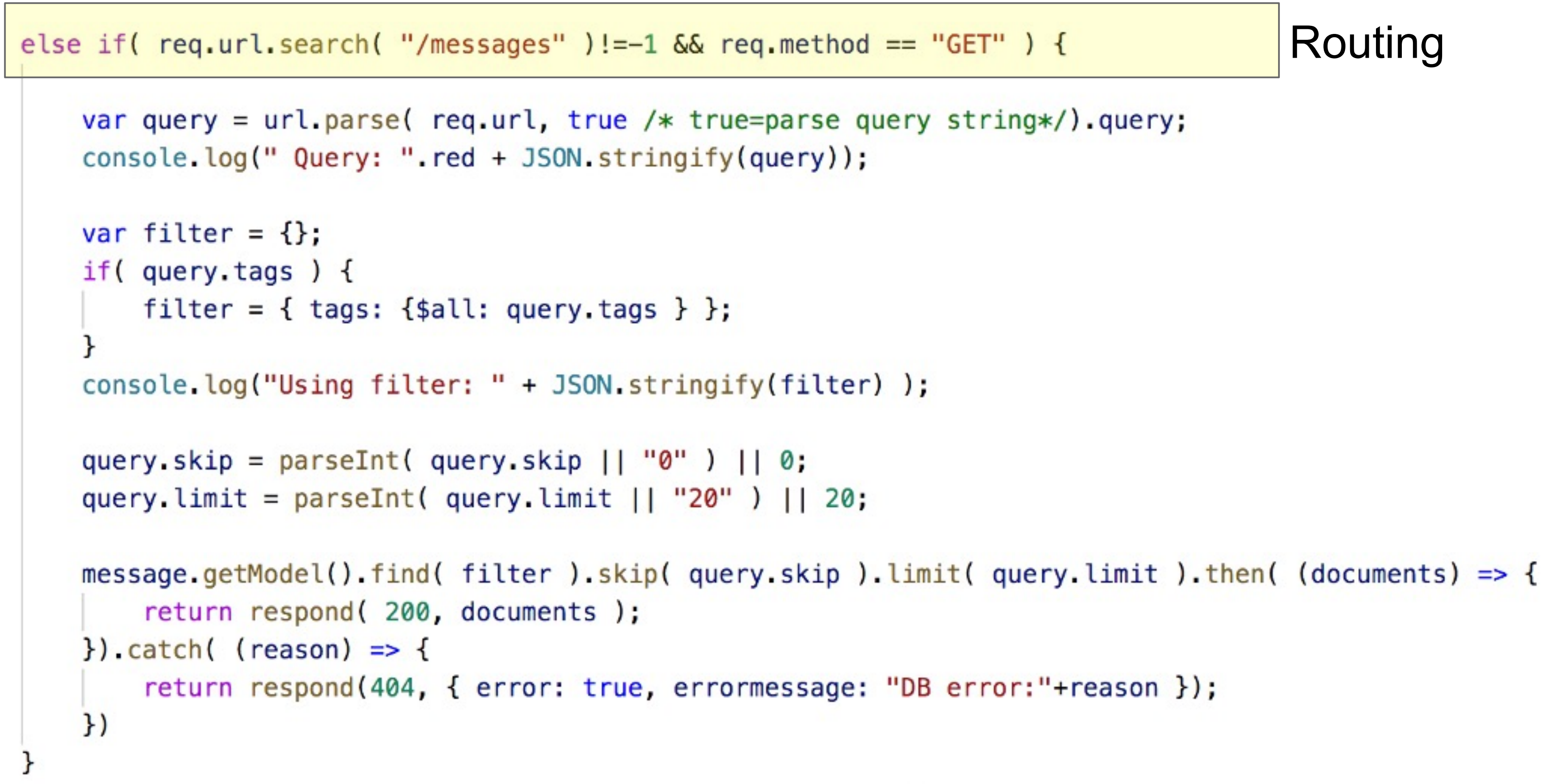
Express is a Node.js module that provides a simple and robust middleware infrastructure for building WEB applications.

It allows us to solve the following problems in a structured way:

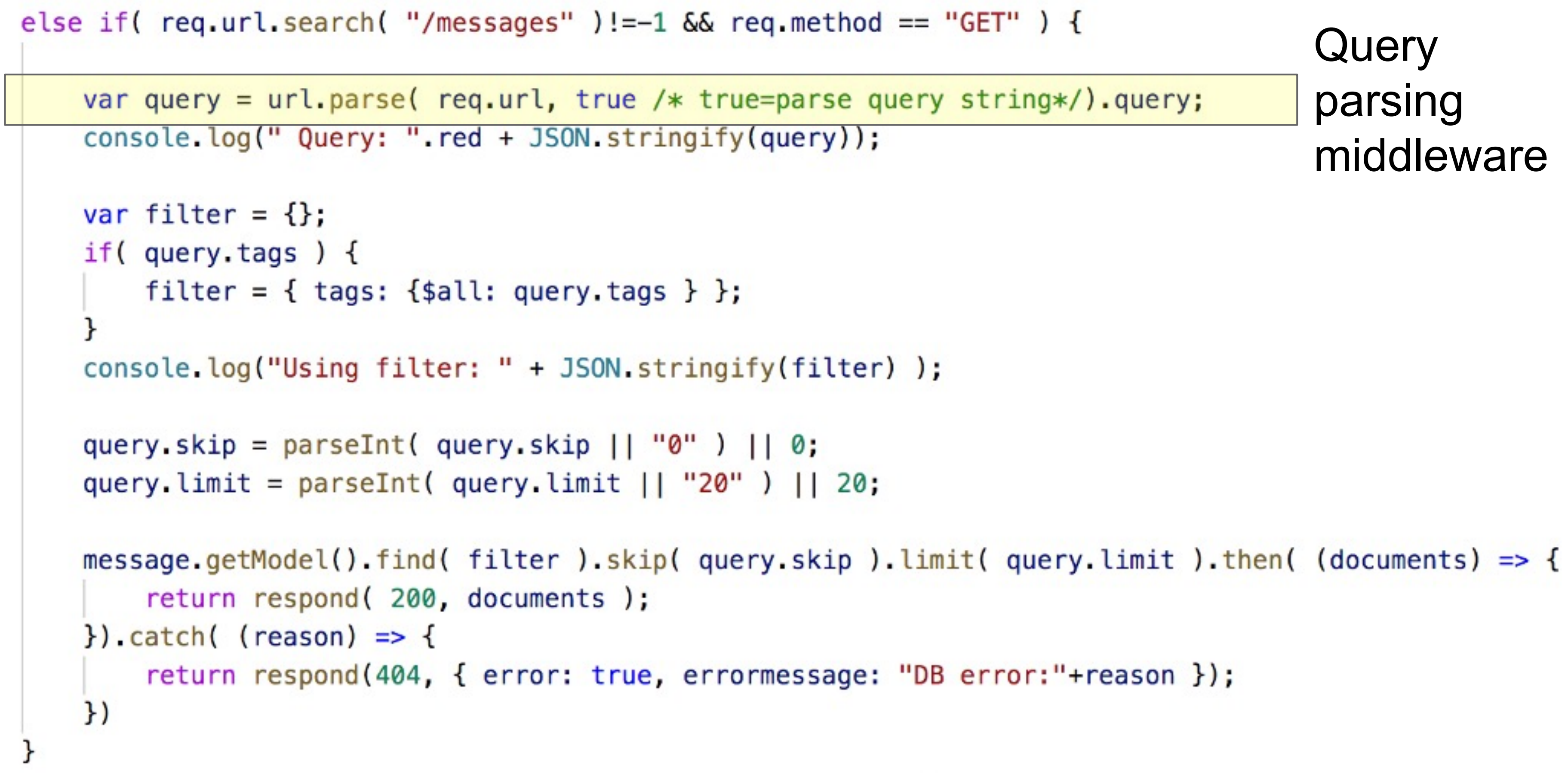
- Have multiple functions that read/modify the request and response objects sequentially
- Manage **routing** between endpoints and APIs

```
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```

Routing

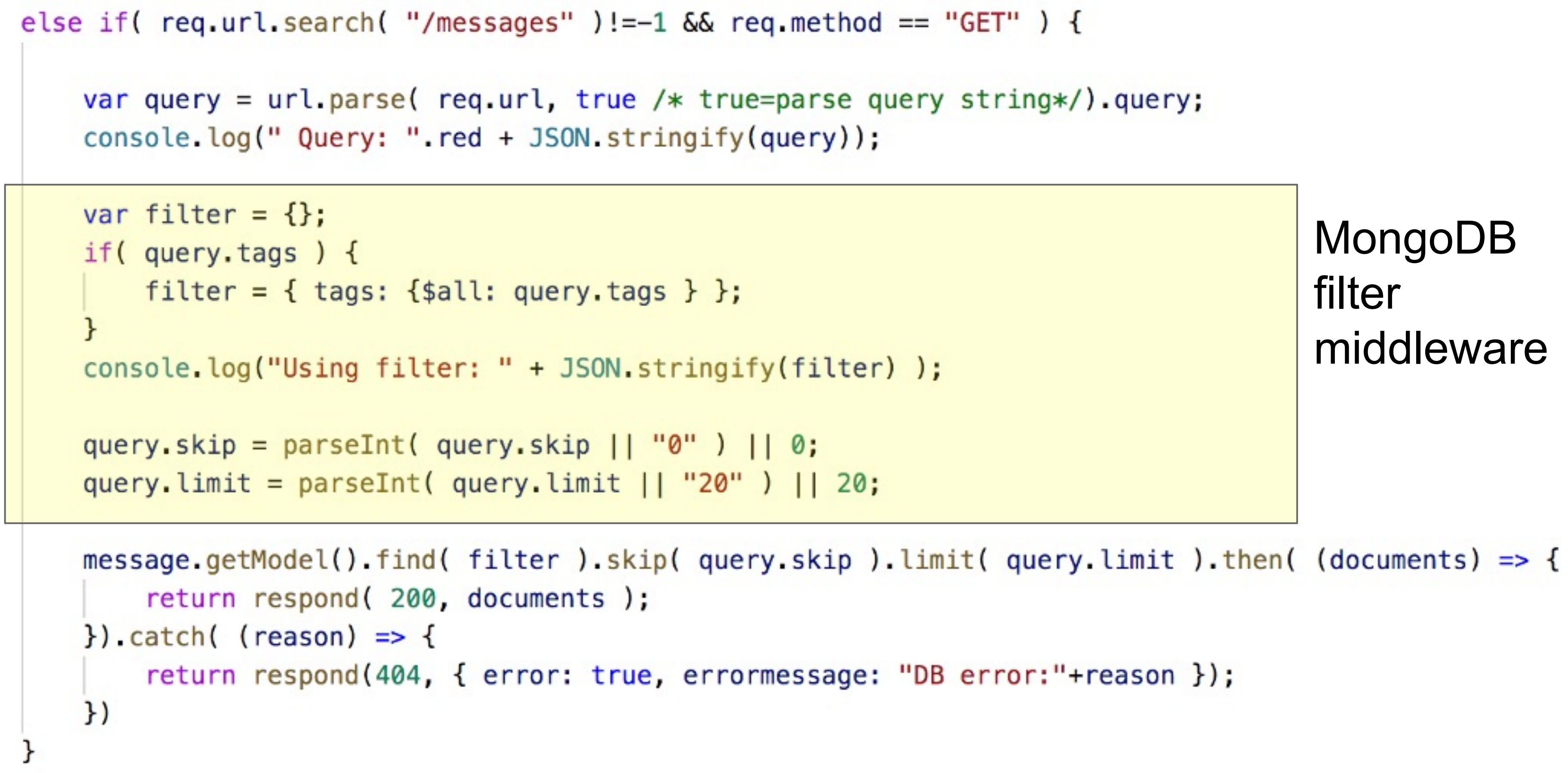


```
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```



```
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```

Query parsing middleware



```javascript
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```

MongoDB
filter
middleware

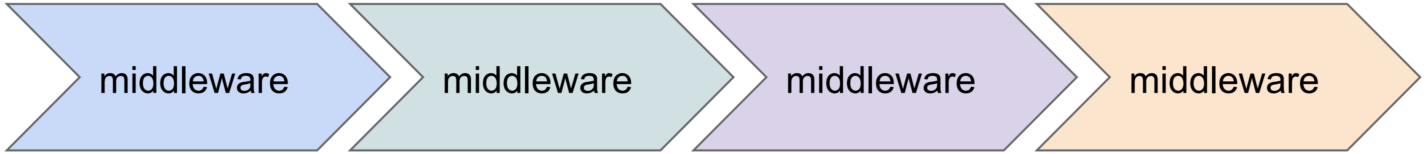
```
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```

Logging middleware

```
else if( req.url.search( "/messages" )!=-1 && req.method == "GET" ) {

    var query = url.parse( req.url, true /* true=parse query string*/).query;
    console.log(" Query: ".red + JSON.stringify(query));

    var filter = {};
    if( query.tags ) {
        filter = { tags: {$all: query.tags } };
    }
    console.log("Using filter: " + JSON.stringify(filter) );

    query.skip = parseInt( query.skip || "0" ) || 0;
    query.limit = parseInt( query.limit || "20" ) || 20;

    message.getModel().find( filter ).skip( query.skip ).limit( query.limit ).then( (documents) => {
        return respond( 200, documents );
    }).catch( (reason) => {
        return respond(404, { error: true, errormessage: "DB error:"+reason });
    })
}
```

Rendering

Middleware



Middleware is a function that takes **req**uest and **res**ponse objects as input, operates on their values, and invokes the **next** middleware function in the pipeline (or ends the response)

```
function (req, res, next) { }
```

built-in middlewares

Express.js contains a set of built-in middlewares to manage the most frequent use cases:

- Parsing of JSON strings and form data
- Data compression
- Insertion and parsing of cookies
- Session management
- Sending of files and static HTML pages

Routing

Routing rules specify how endpoints (URLs) map to their respective middleware pipelines.

Advantages:

- Structured path handling instead of huge if - else if - else blocks
- Use of regular expressions for the matching of a route and the automatic extraction of parameters

Routes definition

```
app.<method>(<path>,[<handlers>])
```

- app is an instance of the Express module
- `<method>` is an HTTP method: get, post, etc.
- `<path>` is the regular expression that indicates the endpoint
- `[<handlers>]` is an array of middleware functions that will be executed in sequence if the request URL matches the `<path>`. Usually, the array is composed of a single function.

Routes defintion

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
})
```

`<path>` can have variable segments to be parsed as parameters (:userId,: bookId).

Express automatically inserts each parameter as properties of the req.params object

Global middleware functions

Some middleware functions can be inserted into the pipeline regardless of the HTTP method or the endpoint path.

Example:

```
app.use( express.bodyParser() )
```

```
app.use( express.methodOverride() )
```

NOTE: Pay attention to the order of insertion!

Error handling middleware

It is possible to define a "special" middleware to manage errors occurring during routing or when executing some other middleware

The Error handling middleware is a function with four arguments, usually installed at the end of the pipeline:

```
function (err, req, res, next) { }
```

Error handling middleware

Error handling functions are invoked:

- If an exception has been thrown (However, exceptions should be avoided in an asynchronous context)
- If a middleware calls next (..) without passing the request and response, but only an object that describes the error **(recommended method)**

Static pages management

One of the most used built-in middleware is for sending static pages to clients:

- Static files are inserted into one or more directories <dir1>, <dir2>, etc
- The middleware is inserted into the pipeline at the desired level:

```
app.use(express.static(‘<dir1>’));
app.use(express.static(‘<dir2>’));
```

More info

- Express.js page: https://expressjs.com/
- Passport middleware for client authentication: https://www.passportjs.org/
- JWT authentication middleware: https://www.npmjs.com/package/express-jwt