

Express Middleware & Routing

Frank Walsh

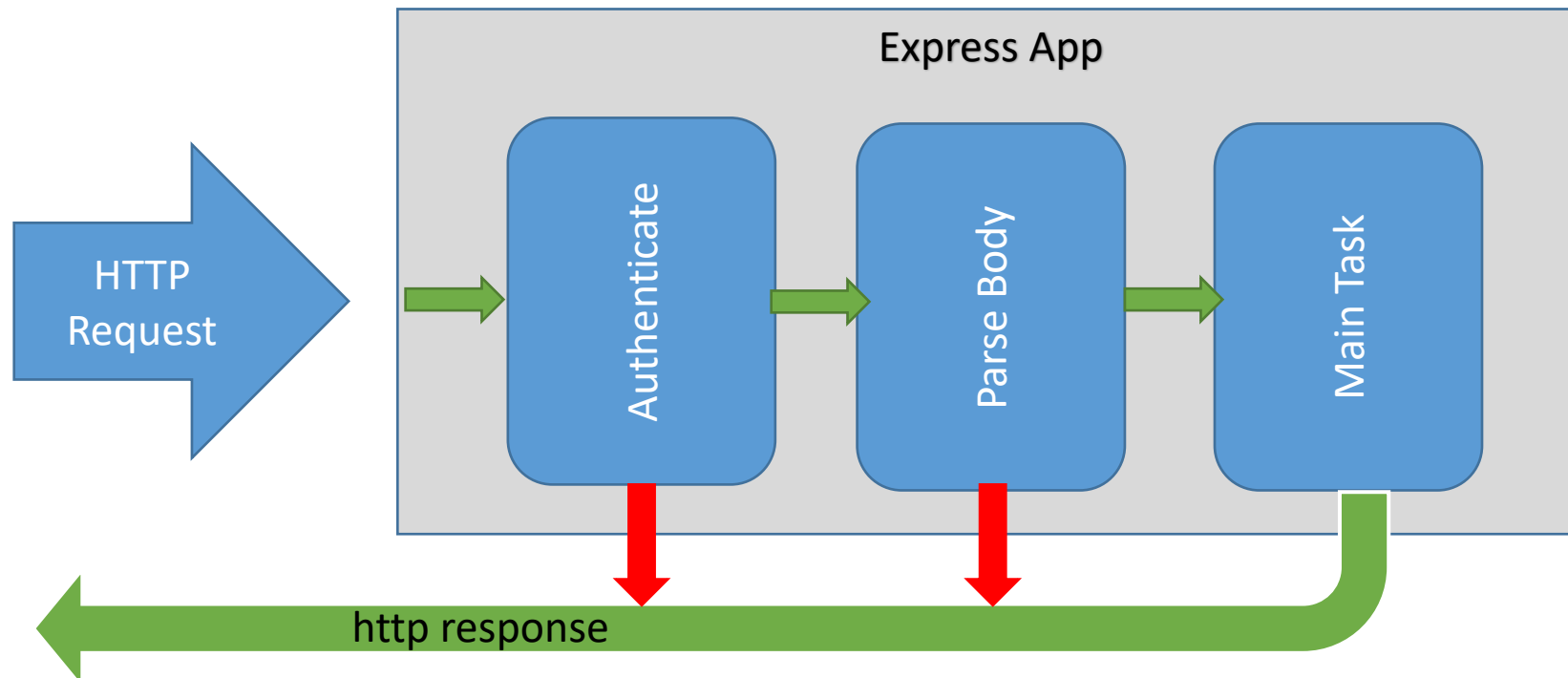
Agenda

- Express Middleware
- Routing in Express
- The Request and Response object

Express Middleware

Express Middleware stack

- The HTTP request (also the response) passes through a pipeline/stack of middleware functions
- Some task is executed at each stage:



Express Middleware

- Middleware functions have access to the request object (req), the response object (res), and the next middleware function in an express application's request-response cycle.
- Your express app will have a 'stack' of middleware functions that can
 - Execute any code.
 - Make changes to the request and the response objects.
 - End the request-response cycle.
 - Call the next middleware function in the stack.

```
11  const middleware1 = (req, res, next) => {  
12    console.log('in middleware 1');  
13    next();  
14  };  
15  
16  app.use(middleware1);  
17  app.use(express.static('public'));
```



Express Middleware Types

- 3rd Party (e.g. [body-parser](#))
- Router level
- App level (app.use(...)) in last lab)
 - Every request is handled
- Error handlers
 - Takes error as first parameter
(err, req, res, next) => { }
- Baked in
 - Express.static()

Middleware Functions

- App level, 3rd party receive 3 arguments

```
const middleware1 = (req,res,next)=>{  
  console.log('in middleware 1');  
  next();  
}
```

- Error Handling middleware receive 4 arguments(error first)

```
const errorHandler1=(err,req,res,next)=>{  
  console.log('in error handler');  
  console.log(err);  
  res.status(500).end('something went wrong!');  
}
```

Express Middleware – Error Middleware

```
11  const middleware1 = (req, res, next) => {
12    console.log('in middleware 1');
13    next(new Error('BOOM!')); // for error handler example
14    // next(); // for general middleware example
15  };
16
17  const errorHandler1 = (err, req, res, next) => {
18    console.log('error handler!!!');
19    console.log(err);
20    next();
21  };
22
23  app.use(middleware1);
24  app.use(express.static('public'));
25  app.use('/api/contacts', contactsRouter);
26  app.use(errorHandler1);
```

Raise error and pass on to next error handling middleware in middleware stack

NOTE: Middleware stack processed in the order it appears in script.

Express Routers

Exports router instance

- Can have several "routers" to implement your APIs.
- Router can have its own routing and middleware
 - Good for multiple APIs/ versioning
- Still uses the application level middleware of express app.

Mount router to URL.
/api/contacts becomes **Base Route** for router

/api/contacts/index.js (contacts router)

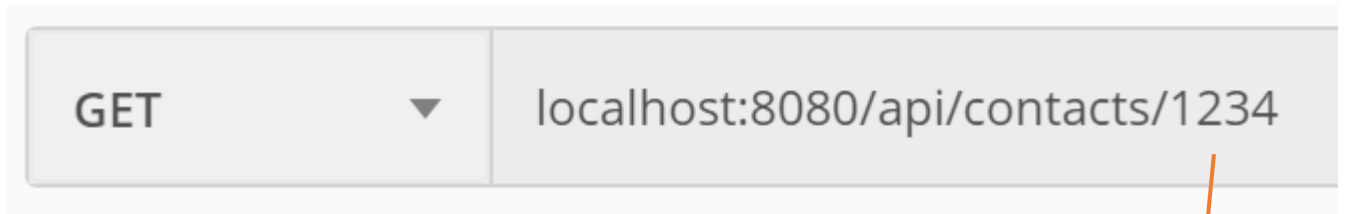
```
1 import express from 'express';
2 import {contacts} from './contacts';
3
4 const router = express.Router(); // eslint-disable-line
5 router.get('/', (req, res) => {
6   res.send({contacts: contacts});
7 });
8
9 export default router;
```

/index.js (express app)

```
1 import dotenv from 'dotenv';
2 import express from 'express';
3 import contactsRouter from './api/contacts';
4
5 dotenv.config();
6
7 const app = express();
8
9
10 app.use(express.static('public'));
11 app.use('/api/contacts', contactsRouter);
12 app.use(errorHandler1);
13
```

Express Routers – Parameters

- Route parameters are named URL segments that capture the values specified at their position in the URL.
- The **req.params** object contains the parameter values, with the name of the route parameter specified in the path as their respective keys.



```
router.get('/:id', (req, res) => {  
  const id = req.params.id; //gets id param from URL  
  //usually retrieve details for customer but for now just return id  
  return res.status(200).end(`id parameter from URL is ${id}`);  
});
```

```
1 id parameter from URL is 1234
```

Express Request Object

- The **req** object represents the HTTP request.
by convention, the object is referred to as '**req**',
Response is '**res**'
- Can use it to access the request query string, parameters, body, HTTP headers.
- Example:

Parameterised URL. Access
using req.params.id

```
router.get('/user/:id',(req, res)=>{  
  res.send('user ' + req.params.id);  
});
```

Express Request Object req.body

- Contains data submitted in the request body.
- Usually need 3rd party body-parsing middleware such as **body-parser**.
- This example shows how to use body-parsing middleware to populate req.body.

index.js code snip (express app)

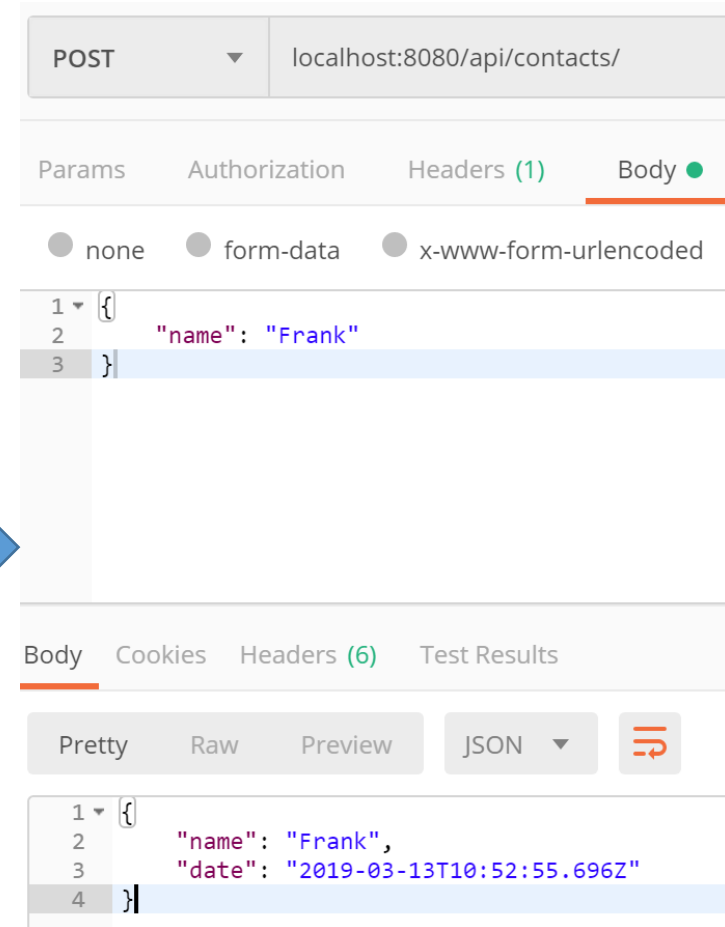
```
// parse application/json
app.use(bodyParser.json())

app.use('/api/contacts', contactsRouter);
```

/api/contacts/index.js code snip (contactsRouter)

```
router.post('/', (req, res) => {
  console.log(req.body);
  req.body.date = new Date();
  // just echo the request json body in the response
  res.json(req.body).end();
});
```

Testing using
Postman



Express Response Object

- The **res** object represents the HTTP response that an Express app sends when it gets an HTTP request.

```
//Add a contact
router.post('/', (req, res) => {
  let newContact = req.body;
  if (newContact){
    contacts.push({name: newContact.name, address : newContact.address});
    res.status(201).send({message: "Contact Created"});
  }else{
    res.status(400).send({message: "Unable to find Contact"});
  }
});
```

Express Response Properties

- **res.send([body])**

- The body parameter can be a String, an object, or an Array.
- For example:

```
res.send({ some: 'json' });  
res.send('<p>some html</p>'); res.status(404).send('Sorry, we cannot find that!');  
res.status(500).send({ error: 'something blew up' });
```

Response Properties

- **res.json([body])**

- actually calls res.send(), but before that it:
 - respects the json spaces and json replacer app setting
 - ensures the response will have utf8 charset and application/json content-type

```
res.json({ user: 'tobi' })
```

```
res.status(500).json({ error: 'message' })
```

Response Properties

- **res.format(object)**

- Performs content-negotiation on the Accept HTTP header on the request object
- Addresses "multiple representations" REST principle

```
res.format({
  'text/plain': function(){
    res.send('hey');
  },

  'text/html': function(){
    res.send('<p>hey</p>');
  },

  'application/json': function(){
    res.send({ message: 'hey' });
  },

  'default': function() {
    // log the request and respond with 406
    res.status(406).send('Not Acceptable');
  }
});
```


Filters

If you want to
authenticate for
access to
resources you can
use multiple
callbacks built into
express routing

Multiple Callbacks

```
function requireLogin(req, res, next) {  
  if (req.session.loggedIn) {  
    next(); // allow the next route to run  
  } else {  
    // require the user to log in  
    res.redirect("/login"); // or render a form, etc.  
  }  
}  
  
// Automatically apply the `requireLogin` middleware to all  
// routes starting with `/admin`  
router.all("/admin/*", requireLogin, (req, res, next)=> {  
  next(); // if the middleware allowed us to get here,  
          // just move on to the next route handler  
});  
  
router.get("/admin/posts", (req, res)=> {  
  // if we got here, the `app.all` call above has already  
  // ensured that the user is logged in  
});
```

Express Middleware

- Express is a Routing and Middleware framework.
 - You've seen the routing in the previous lab
- Middleware functions have access to the Request, Response and the **next()** function
 - The next function calls the next middleware function.
- Use middleware to
 - Change the request/response
 - End the request/response cycle
 - Call the next middleware in the stack.
- If middleware does not call next() or return, express will just hang
 - Can be an issue with promises but can be resolved

```
11  const middleware1 = (req, res, next) => {  
12    console.log('in middleware 1');  
13    next();  
14  };  
15  
16  app.use(middleware1);  
17  app.use(express.static('public'));
```