MVC-based JavaScript Web App

Outline

1. Big Picture
2. MVC-based Web applications
3. Node.js Express Framework
4. View Template using Handlebars
5. Server-side Rendering of Views

Web Client

Request

Response

Web Server

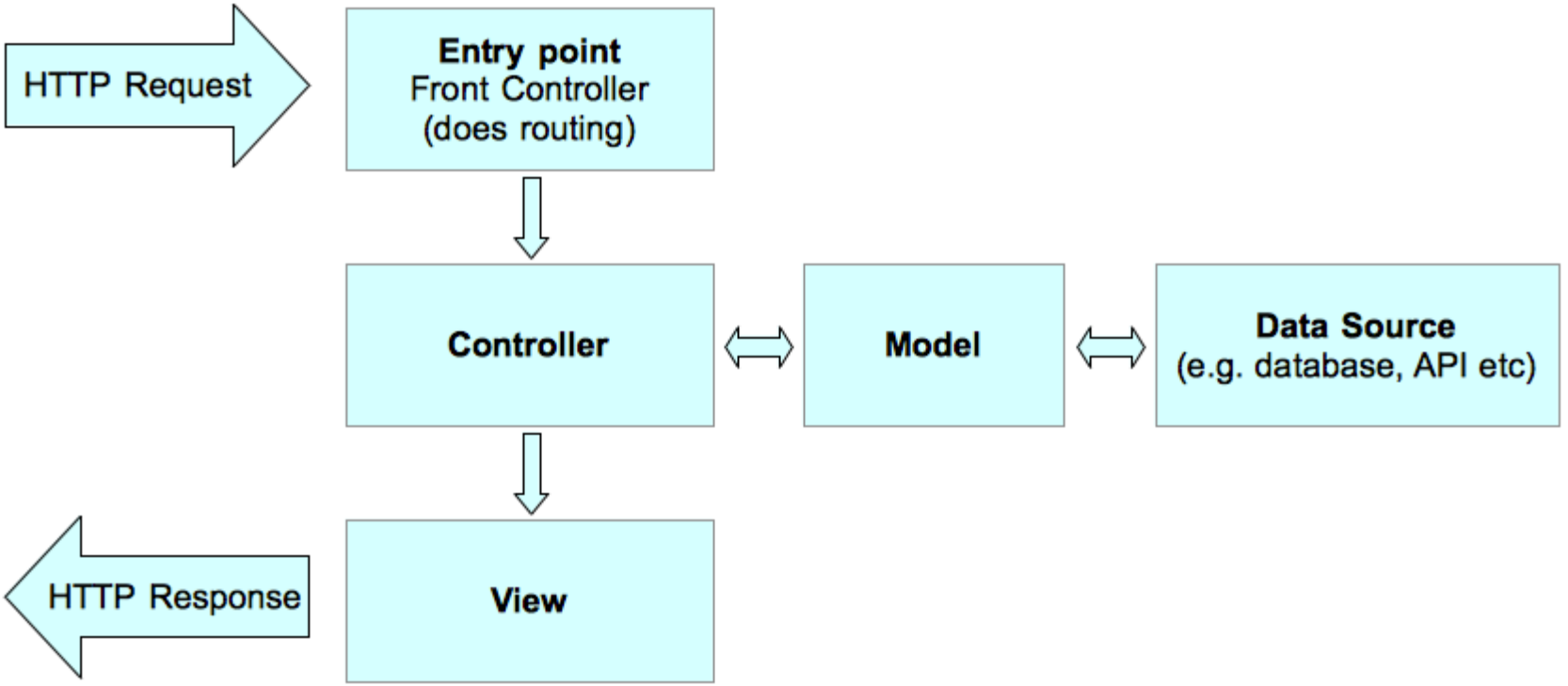
Frontend development

- HTML for page structure
- CSS for styling
- JavaScript for interaction
- AJAX for partial page updates (without reload)

Backend development

- Web API
- Views Template
- Data storage

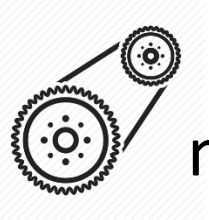
MVC-based Web applications



MVC-based Web application

Controller

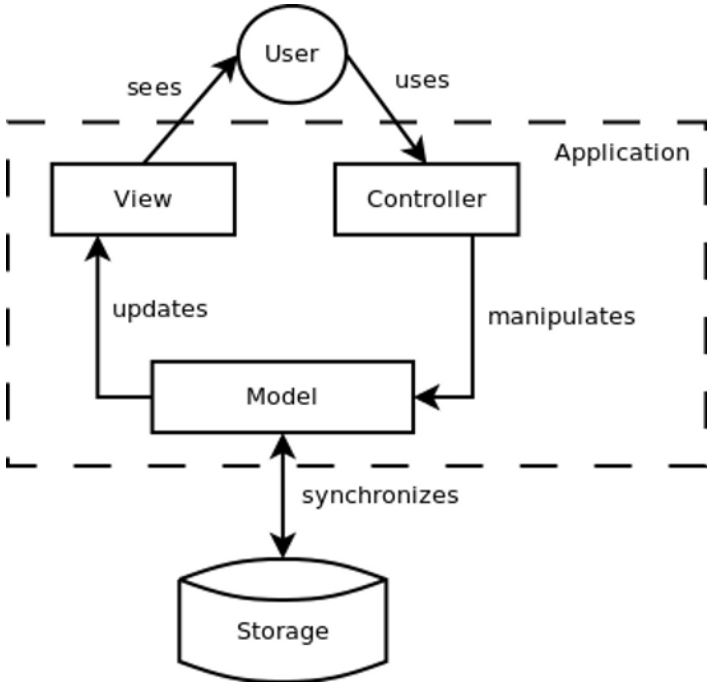
- accepts incoming requests and user input and **coordinates** request handling
- instructs the model to perform actions based on that input
  - e.g. add an item to the user's shopping cart
- decides what view to display for output



Model : implements business logic and computation, and manages application's data

View : responsible for

- collecting input from the user
- displaying output to the user

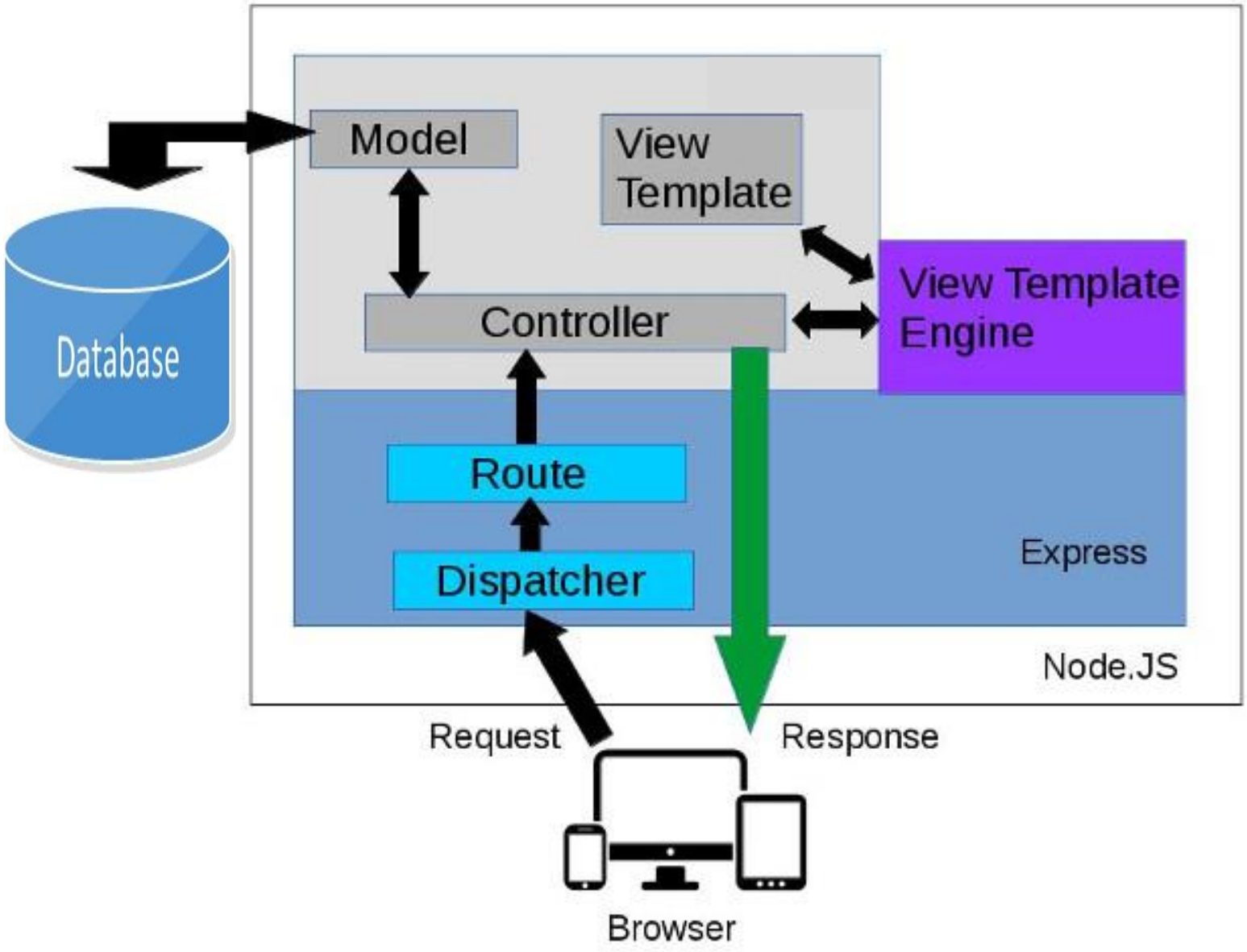


Advantages of MVC

- ***Separation of concerns***
  - Views, controller, and model are separate components. This allows modification and change in each component without significantly disturbing the others.
    - Computation is not intermixed with Presentation. Consequently, code is cleaner and easier to understand and change.
- **Flexibility**
  - The view component, which often needs changes and updates to keep the users continued interests, is separate
    - The UI can be completely changed without touching the model in any way
- **Reusability**
  - The same model can used by different views (e.g., Web view and mobile view)
- **Allows for parallel work**, e.g., a UI designer can work on the View while a software engineer works on the Controller and Model

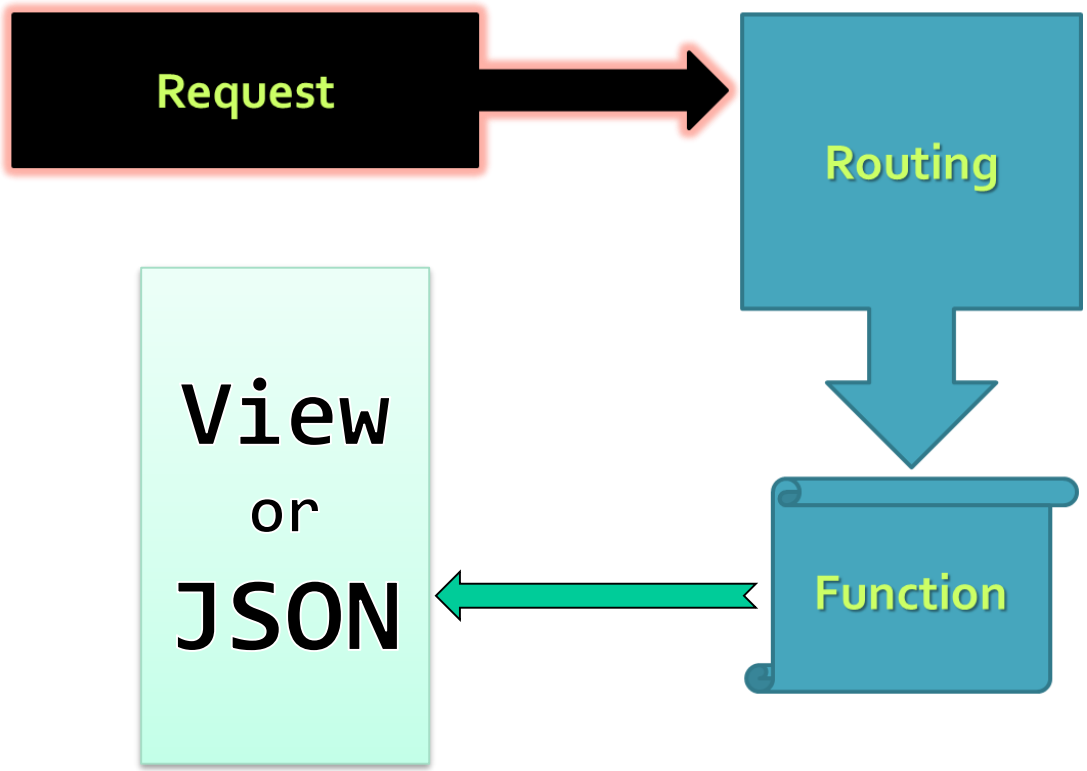
MVC is widely used and recommended particularly for interactive web-applications

MVC using Node.js Express

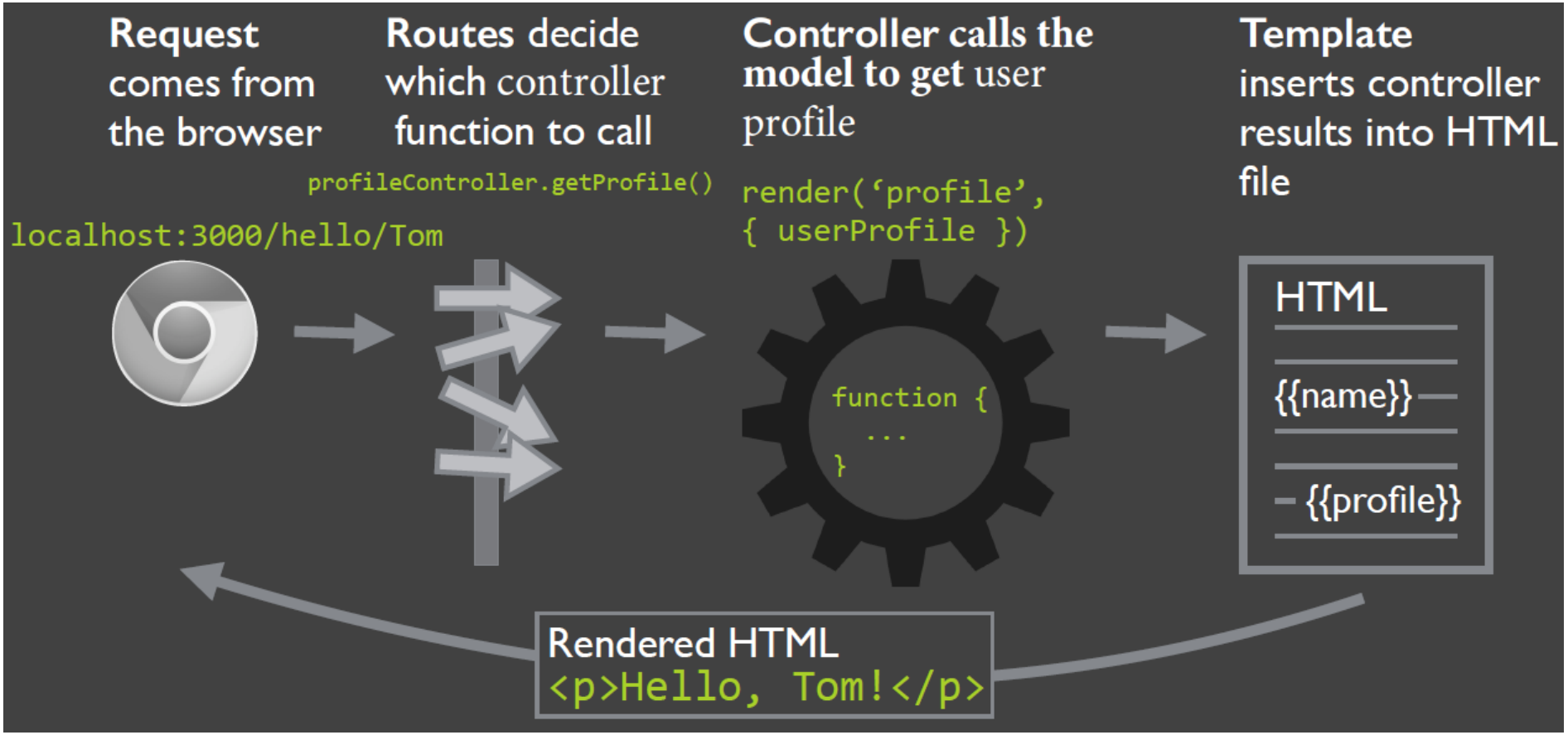


Express

Web Application Framework for Node.js



Interaction between App Modules



Create and Start an Express App

```
let express = require('express')
let app = express()

app.get('/', (req, res) => {
    res.send('السلام عليكم ورحمة الله وبركاته')
})

let port = 3000
app.listen(port, () => {
    console.log(`App is available @ http://localhost:${port}`)
})
```

- A function **registered** to listen to the URL http://localhost:3000/
- When someone visits this Url the function associated with get '/' will run and 'السلام عليكم ورحمة الله وبركاته' will be returned to the requester

Routing

- Routing is a way to **map** of an HTTP verb (like GET or POST) and a URI (like /users/123) to a **handler**

- To a receive a **query string**, a **parameter** can be added to the route uri with a colon in front of it. To grab the value, you'll use the **params** property of the request

```
app.get('/api/students/:id', (req, res) => {
    let studentId = req.params.id
    console.log('req.params.id', studentId)
})
```


11

Route Parameters

- Route parameters are **named** URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the **req.params** object

```
app.get('/authors/:authorId/books/:bookId', (req, res) => {
    // If the Request URL was http://localhost:3000/authors/34/books/8989
    // Then req.params: { authorId: "34", bookId: "8989" }
    res.send(req.params)
})
```


12

Express Router

- For simple app routes can defined in app.js
- For large application, Express Router allows defining the routes in a separate file(s) then attaching routes to the app to:
  - Keep app.js clean, simple and organized
  - Easily find and maintain routes

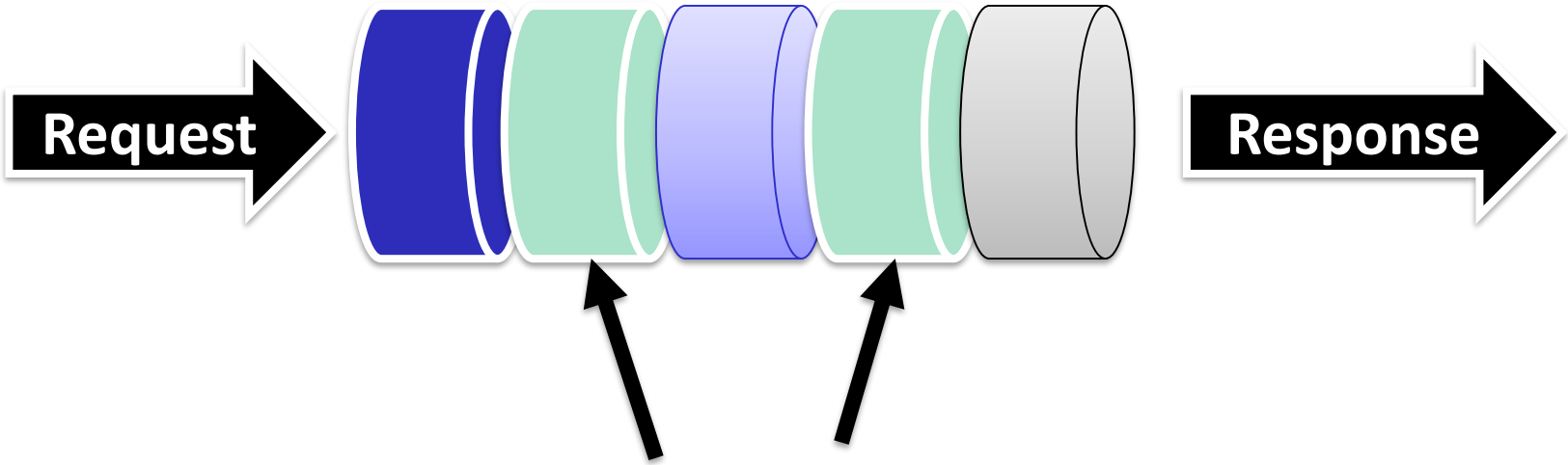
```
// routes.js file
let router = express.Router()
router.get('/api/students', studentController.getStudents )
module.exports = router

//app.js file - mount the routes to the app
let routes = require('./routes')
app.use('/', routes)
```

Express Middleware

- Express middleware allows you to **pipeline** a single request through a series of functions.
- Request Processing Pipeline: the request passes through an *array* of functions before it reaches your route handler. e.g.,

```
/* body-parser extracts URL encoded text from the
body of the incoming request and assigns it to req.body */
app.use( bodyParser.urlencoded({extended: true}) )
```

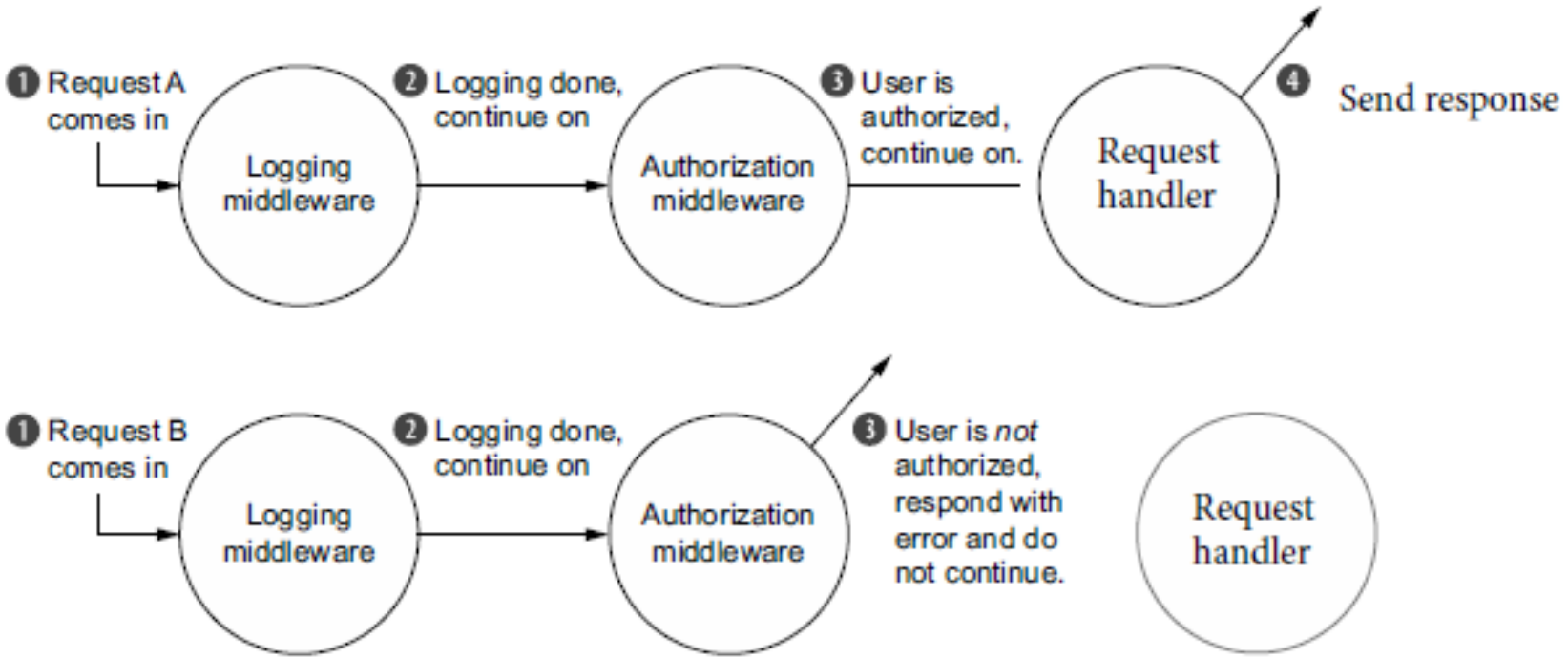


Middleware (bodyParser, logging, authentication, router etc.)

Middleware Example

- Each middleware function may modify the request or the response. This modularity make it easier to use and compose existing middleware packages such the middleware for serving static files

```
//Allow serving static files from __dirname which is the current folder)
app.use( express.static(__dirname) )
```



Example BodyParser middleware

```
<h2>Login</h2>
<form method="post" action="/">
  Username: <input type="text" name="username" />
  Password: <input type="password" name="password" />
  <input type="submit" value="Submit" />
</form>
```

```
const bodyParser   =  require('body-parser')
/* body-parser extracts URL encoded text from the
body of the incoming request and assigns it to req.body */
app.use( bodyParser.urlencoded( {extended: true}) )

app.post('/', (req, res) => {
    console.log(req.body)
    res.send('Welcome ' + req.body.username)
})
```

Custom Middleware Example

```
let express = require('express')
let app = express()

//Define a middleware function
function logger (req, res, next) {
    req.requestTime = new Date()
    console.log(`Request received at ${req.requestTime}`)
    next()
}

// Attach it to the app
app.use(logger)

app.get('/', function (req, res) {
    let responseText = `Hello World!<br>
        Requested at: ${req.requestTime}`
    res.send(responseText)
})

let port = 3000
app.listen(port, () => {
    let host = "localhost"
    console.log(`App is running and available @ http://${host}:${port}`)
})
```

Views

Template using Handlebars

http://handlebarsjs.com/

View Template

- **View template** used to dynamically generate HTML pages on-demand based on user input
- **View engine** (template engine) is a library that generates HTML page based on a template and a given JavaScript object
  - Provide cleaner solution by separating the view
- There are lots of JavaScript view engines such as Handlebars.js, KendoUI, Jade, Angular, etc.
- **Handlebars.js** is recommended. It is a library for creating client-side or server-side UI templates

Usage

- Add Handlebars script

```
<script src="path/to/handlebars.js"></script>
```

- Create a template

```
let studentTemplate =`
    <table>
         <tr>
            <td>StudentId</td>
            <td>{{id}}</td>
        </tr>
        <tr>
            <td>Name</td>
            <td>{{firstname}} {{lastname}}</td>
        </tr>
     </table>`
```

Template variable to be replaced with a value that is passed to the template

- Render the template

```
let student = {id: '…', firstname: '…', lastname: '…'},
    htmlTemplate = Handlebars.compile(studentTemplate)
studentDetails.innerHTML = htmlTemplate(student)
```

Creating HTML Templates

- HTML template has **placeholders** that will be replace by **data** passed to the template
- Handlebars.js marks placeholders with double curly brackets `{{value}}`
  - When rendered, the placeholders between the curly brackets are replaced with the corresponding value

Iterating over a list of elements

- **{{#list}} {{/list}}** block expression is used to iterate over a list of objects
  - Everything in between will be evaluated for each object in the collection

```
<select id="studentsDD">
    <option value=""></option>
    {{#students}}
        <option value="{{studentId}}">
              {{studentId}} - {{firstname}} {{lastname}}
        </option>
     {{/students}}
</select>
```

```
let students = [{
    "studentId": 2015001,
    "firstname": "Fn1",
    "lastname": "Ln1"
  },
  {
    "studentId": 2015002,
    "firstname": "Fn2",
    "lastname": "Ln2"
  }]
```

Conditional Expressions

- Render fragment only if a property is true
  - Using `{{#if property}} {{/if}}`
    or `{{unless property}} {{/unless}}`

```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{else}}
    <h1>Unknown Author</h1>
  {{/if}}
</div>
```

```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```

The with Block Helper

- `{{#with obj}} {{/with}}`
  - Used to minify the path
  - Write `{{prop}}` Instead of `{{obj.prop}}`

```
<div class="entry">
  <h1>{{title}}</h1>

  {{#with author}}
  <h2>By {{firstName}} {{lastName}}</h2>
  {{/with}}
</div>
```

```
{
  title: "My first post!",
  author: {
    firstName: "Abbas",
    lastName: "Ibn Farnas"
  }
}
```

Server-side Rendering of Views

Client-side vs. Server-side Rendering of Views

- Client-side Views Rendering **frees the server** from this burden and enhances scalability
  - But one of the main disadvantages is slower initial loading speed as the client receive a lot of JavaScript files to handle views rendering
- Views could be generated on the server side to reduce the amount of client-side JavaScript and speed-up initial page loads particularly for slow clients but this puts the rending burden on the server
  - Web servers may render the page faster than a client side rendering. As a result, the initial loading is quicker.

Configure Handlebars View Engine

```
let handlebars    =  require('express-handlebars')
let app            = express()

/* Configure handlebars:
 set extension to .hbs so handlebars knows what to look for
 set the defaultLayout to 'main'
 the main.hbs defines define page elements such as the menu
 and imports all the common css and javascript files
*/
app.engine('hbs', handlebars({ defaultLayout: 'main',
  extname: '.hbs'}))

// Register handlebars as our view engine as the view engine
app.set('view engine', 'hbs')

//Set the location of the view templates
app.set('views', __dirname + '/views')
```

res.render

- Call **res.render** method to perform server-side rendering and return the generated html to the client

```
res.render('shopCart', { shoppingCart })
```

The above example passes the shopping cart to the **'shopCart'** template to generate the html to be returned to the browser

Resources

- NodeSchool

https://nodeschool.io/

- Mozilla Developer Network

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs

- Learn Handlebars in 10 Minutes
  http://tutorialzine.com/2015/01/learn-handlebars-in-10-minutes/

- JavaScript Standard Style

https://github.com/feross/standard