Κεφάλαιο 12. Ανάπτυξη εφαρμογών με το πακέτο Express.js

Σύνοψη

Στο κεφάλαιο αυτό θα εξετάσουμε το Express.js, ένα πακέτο της Node.js το οποίο παρέχει ένα απλό και ισχυρό πλαίσιο για την ανάπτυξη εφαρμογών διαδικτύου. Θα παρουσιάσουμε τον κύκλο επεξεργασίας ενός αιτήματος πελάτη προς τον εξυπηρετητή και τον τρόπο λειτουργίας της γραμμής επεξεργασίας (middleware pipeline) του Express.js, καθώς και τη δρομολόγηση και την πρόσβαση στα αντικείμενα αιτήματος και απόκρισης. Στη συνέχεια θα χρησιμοποιήσουμε τη μηχανή template Handlebars για την προετοιμασία της απάντησης.

Προαπαιτούμενη γνώση

Για τη μελέτη αυτού του κεφαλαίου είναι απαραίτητη η εξοικείωση με τη Node.js, που παρουσιάστηκε στο κεφάλαιο 11 καθώς και με την JavaScript (κεφάλαια 7, 8, 9 και 10). Επίσης, για την κατασκευή templates χρησιμοποιείται, σε βασικό επίπεδο μόνο, και η γλώσσα HTML που παρουσιάζεται στα κεφάλαια 2 και 3.

12.1 Εισαγωγή

Το Express.js είναι μια βιβλιοθήκη της Node.js που μας βοηθά να αναπτύξουμε εφαρμογές Node.js στην πλευρά του εξυπηρετητή. Αν και θα μπορούσαμε να χρησιμοποιήσουμε απευθείας τη Node.js για να γράψουμε τις εφαρμογές μας, ωστόσο οτιδήποτε ξεπερνά τα όρια ενός βασικού παραδείγματος γρήγορα μπορεί να μας οδηγήσει να γράφουμε κώδικα για εργασίες που είναι κοινές στις περισσότερες εφαρμογές διαδικτύου και για τις οποίες έχουν δοθεί λύσεις. Αυτές τις λύσεις μπορούμε να τις χρησιμοποιήσουμε έτοιμες ή, τουλάχιστον, μέσα σε ένα πλαίσιο που θα βοηθήσει στην ανάπτυξη και τη συντήρηση ή επέκταση της εφαρμογής μας.

Μερικές τέτοιες βασικές εργασίες με τις οποίες διευκολυνόμαστε είναι ενδεικτικά:

1. HTTP Request parsing, Cookie parsing
2. Session management
3. Request routing
4. Send HTTP Response
5. Error handling

12.2 Μια απλή εφαρμογή με το Express.js

Το Express.js είναι ένα module της Node.js και εγκαθίσταται με τον συνήθη τρόπο. Για να κατασκευάσουμε μια εφαρμογή με το Express.js, αρχικοποιούμε, αν δεν το έχουμε ήδη κάνει, το πρότζεκτ μας:

```
npm init -y
```

Στη συνέχεια εγκαθιστούμε το πακέτο express και τις εξαρτήσεις του:

```
npm install express
```

Σε ένα αρχείο με όνομα, για παράδειγμα, index.mjs γράφουμε μια υποτυπώδη εφαρμογή διαδικτύου:

```
import express  from 'express'
const app = express()

app.get('/', (req, res) => res.send('Γεια σου express!'))
app.listen(3000, () => console.log('Η εφαρμογή τρέχει'))
```

Ξεκινάμε την εφαρμογή με τον τρόπο που το κάνουμε για όλες τις εφαρμογές διαδικτύου στη Node.js, γράφοντας δηλαδή στο τερματικό node index.mjs ή προσθέτοντας το κατάλληλο σκριπτ στο package.json ή με το πακέτο nodemon, όπως είδαμε στο προηγούμενο κεφάλαιο. Η εφαρμογή μας τρέχει και μπορούμε να τη δούμε στον φυλλομετρητή στη σελίδα http://localhost:3000.

12.3 Ο κύκλος επεξεργασίας του αιτήματος

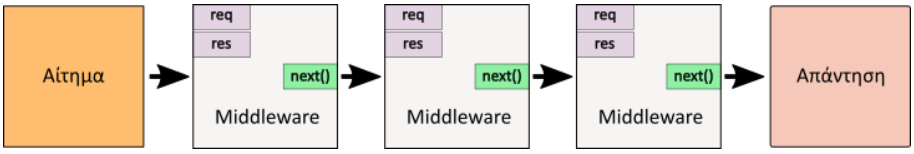
Το Express.js χρησιμοποιεί εκτεταμένα το middleware pattern. Με την προσέγγιση αυτή, κάθε αίτημα που δέχεται η εφαρμογή περνάει από διαδοχικά υποτμήματα (middleware), το ένα μετά το άλλο, που επεξεργάζονται το αίτημα μέχρι να σχηματιστεί και να αποσταλεί η απόκριση στον πελάτη. Μια εφαρμογή Express.js αποτελείται, λοιπόν, από επιμέρους υποτμήματα (τα middleware), τα οποία συντίθενται σε μια αλυσίδα επεξεργασίας (**Εικόνα *12.1***), που στο Express.js ονομάζεται "pipeline". Η αλυσίδα επεξεργασίας είναι μια απλή αλλά πολύ ισχυρή ιδέα. Καθώς πολλές εργασίες στις εφαρμογές διαδικτύου είναι κοινές σε μεγάλο πλήθος εφαρμογών, μπορούμε να χρησιμοποιήσουμε ένα μεγάλο πλήθος από έτοιμα πακέτα που παρέχουν εξειδικευμένα middleware.

12.3.1 Τα middleware του Express.js

Στη αλυσίδα επεξεργασίας κάθε υποτμήμα της (middleware) δέχεται δύο αντικείμενα, το **αντικείμενο αιτήματος** (request object), που περιέχει πληροφορίες για το αίτημα του πελάτη και το **αντικείμενο απόκρισης** (response object), που περιέχει την απόκριση που θα στείλει στο τέλος ο εξυπηρετητής στον πελάτη. Ένα middleware είναι μια απλή συνάρτηση της JavaScript που μπορεί να εκτελέσει οποιαδήποτε επεξεργασία, ακριβώς όπως μια συνήθης συνάρτηση. Μπορεί να διαβάσει και να επεξεργαστεί τα αντικείμενα του αιτήματος και της απόκρισης και, επιπλέον, επειδή είναι μέρος της αλυσίδας επεξεργασίας, όταν το υποτμήμα τελειώσει με την επεξεργασία του, πρέπει να κάνει ένα από τα δύο:

1. να δώσει τον έλεγχο στο επόμενο υποτμήμα της αλυσίδας επεξεργασίας ή
2. να τερματίσει την αλυσίδα επεξεργασίας στέλνοντας μια απόκριση στο αίτημα του πελάτη.

Αν δεν γίνει κάτι από τα δύο, π.χ. λόγω κάποιου σφάλματος, το αίτημα του πελάτη θα μείνει αναπάντητο και θα εκκρεμεί.



Εικόνα 12.1 *Η αλυσίδα επεξεργασίας του Express.js.*

Γενικότερα, ένα υποτμήμα της αλυσίδας επεξεργασίας

1. επεξεργάζεται και μεταβάλλει τα αντικείμενα request και response και διεκπεραιώνει κάποιες ενέργειες, για παράδειγμα,
    1. καταγραφή συμβάντων (logging),
    2. αυθεντικοποίηση (authentication),
    3. επεξεργασία του αιτήματος κ.λπ.
2. τερματίζει τη λειτουργία του με έναν από τους δύο τρόπους:
    1. προωθεί το request και το response object στο επόμενο υποτμήμα,
    2. αποκρίνεται στο αίτημα του πελάτη, δηλαδή τερματίζει την αλυσίδα επεξεργασίας.

Ένα middleware είναι μια συνάρτηση με τρία ορίσματα (`req`, `res`, `next`), π.χ.

```
const myMiddleware = (req, res, next) => {
    console.log('my middleware')
    next()
}
```

Οι παράμετροι ονομάζονται συμβατικά req, res, και next. Τα req και res είναι τα αντικείμενα του αιτήματος και της απάντησης, όπως ακριβώς και στη Node.js (Ενότητα 11.6). Η παράμετρος next είναι το επόμενο υποτμήμα στην αλυσίδα επεξεργασίας. Στο παράδειγμα, η `next()` είναι η συνάρτηση που το myMiddleware θα καλέσει όταν διεκπεραιώσει τον ρόλο του στην αλυσίδα επεξεργασίας.

12.3.2 Καθορισμός της αλυσίδας επεξεργασίας

Μπορούμε να καθορίσουμε τα ενδιάμεσα βήματα της αλυσίδας επεξεργασίας με τη συνάρτηση

app.use(middleware). Η σειρά με την οποία καλείται η app.use() καθορίζει και τη σειρά των τμημάτων στην αλυσίδα επεξεργασίας.

Στο παρακάτω παράδειγμα έχουμε κατασκευάσει δύο απλοϊκά υποτμήματα, που το μόνο που κάνουν είναι καταγράφουν ότι καλέστηκαν. Στη συνέχεια, με την app.use() προσδιορίζουμε τη σειρά με την οποία θα κληθούν στην αλυσίδα επεξεργασίας, πρώτο το myMiddleware2, έπειτα το myMiddleware1 και τέλος το myMiddleware3.

```
import express  from 'express'
const app = express()

const myMiddleware1 = (req, res, next) => {
    console.log('MW-1')
    next()
}

const myMiddleware2 = (req, res, next) => {
    console.log('MW-2')
    next()
}

const myMiddleware3 = (req, res, next) => {
    console.log('MW-3')
    res.send("Ok!")
}

app.use(myMiddleware2)
app.use(myMiddleware1)
app.use(myMiddleware3)
```

Η χρήση της app.use() στο πάνω παράδειγμα σημαίνει και ότι αυτά τα υποτμήματα θα εκτελεστούν σε **κάθε** αίτημα HTTP. Στο τερματικό θα εμφανιστούν τα μηνύματα με αυτή τη σειρά, ανεξάρτητα από τι ζήτησε ο χρήστης στο URL:

```
MW-2
MW-1
MW-3
```

Την ίδια σειρά επεξεργασίας μπορούμε εναλλακτικά να ορίσουμε περνώντας μια σειρά από υποτμήματα σαν παραμέτρους στην app.use():

```
app.use(myMiddleware2, myMiddleware1, myMiddleware3)
```

ή χρησιμοποιώντας, για ευκολία, έναν πίνακα:

```
const pipeline = [myMiddleware2, myMiddleware1, myMiddleware3]
app.use(pipeline)
```

Στο υποτμήμα myMiddleware3 τερματίζουμε την αλυσίδα επεξεργασίας με κλήση στη συνάρτηση res.send(). Η send() είναι συνάρτηση του αντικείμενου απόκρισης res η οποία αποστέλλει την απάντηση στον πελάτη και τερματίζει την τρέχουσα αλυσίδα επεξεργασίας. Αν δηλαδή μετακινήσουμε τη myMiddleware3 πιο πριν στην αλυσίδα επεξεργασίας, τα επόμενα υποτμήματα δεν θα εκτελεστούν, καθώς η αλυσίδα θα τερματίσει με την κλήση στη res.send().

Είναι φανερό πως θα ήταν πολύ χρήσιμη δυνατότητα η αλυσίδα επεξεργασίας να μπορεί να είναι διαφορετική για αιτήματα του χρήστη που γίνονται σε διαφορετική διαδρομή ή με διαφορετική μέθοδο. Η διαδικασία αυτή ονομάζεται δρομολόγηση (routing) και είναι πολύ ευέλικτη με το Express.js.

12.3.3 Δρομολόγηση

Η δρομολόγηση καθορίζει τον τρόπο που το Express.js θα αποκριθεί σε αιτήματα πελατών που γίνονται σε

διαφορετικές διαδρομές ή και με διαφορετικές μεθόδους HTTP. Μια διαδρομή (route) καθορίζεται, λοιπόν, από:

1. τη μέθοδο HTTP που χρησιμοποιήθηκε.
2. το μονοπάτι του αιτήματος που έλαβε η εφαρμογή μας.

Η δρομολόγηση μπορεί να γίνει με το αντικείμενο εφαρμογής του Express.js. Αυτό συνήθως ονομάζεται "app" και δημιουργείται με κλήση στον δημιουργό του Express.js: **const** app = express(). Η δρομολόγηση μπορεί όμως να γίνει και με την κλάση Router, που έχει σχεδόν τις ίδιες δυνατότητες δρομολόγησης με το αντικείμενο app και συνήθως προτιμάται, καθώς με τη Router έχουμε τη δυνατότητα να ορίσουμε πολλούς δρομολογητές και να έχουμε πιο σπονδυλωτό κώδικα. Ας δούμε αρχικά τη δρομολόγηση με το αντικείμενο εφαρμογής του Express, που είναι και η πιο απλή περίπτωση.

12.3.4 Προσάρτηση σε συγκεκριμένο μονοπάτι

Είναι πολύ χρήσιμο η εφαρμογή μας να αποκρίνεται με διαφορετικό τρόπο σε διαφορετικά μονοπάτια (path), π.χ. ένα αίτημα στο μονοπάτι "/login" να οδηγεί σε μια σελίδα όπου ο χρήστης θα δώσει τα στοιχεία ταυτοποίησης ή ένα αίτημα στο μονοπάτι "/getUserProfile/maria" να επιστρέψει το προφίλ του χρήστη "maria" κ.ο.κ., δηλαδή η αλυσίδα επεξεργασίας να διαφοροποιείται ανάλογα με το μονοπάτι.

Για να προσαρτήσουμε υποτμήματα σε συγκεκριμένα μονοπάτια, μπορούμε να περάσουμε το μονοπάτι σαν πρώτο όρισμα στην `app.use()`. Στο παρακάτω παράδειγμα έχουμε ορίσει δύο μονοπάτια, το "/path1" και το "/".

```
app.use("/path1", myMiddleware1)
app.use("/path2", myMiddleware2)
app.use(myMiddleware3)
```

Με τον τρόπο αυτό ορίσαμε δύο διαφορετικές αλυσίδες επεξεργασίας για τα δύο μονοπάτια:

1. myMiddleware1 → myMiddleware3, για αιτήματα στο μονοπάτι "/path1",
2. myMiddleware2 → myMiddleware3, για αιτήματα στο μονοπάτι "/path2".

12.3.5 Προσάρτηση σε συγκεκριμένη μέθοδο

Εκτός από το μονοπάτι, μπορούμε να προσδιορίσουμε σαν σημείο προσάρτησης και τη μέθοδο HTTP με την οποία γίνεται το αίτημα του πελάτη. Αν ξαναγράψουμε το προηγούμενο παράδειγμα και αντικαταστήσουμε την `use()` με την `get()`, τότε η αλυσίδα επεξεργασίας θα ισχύει μόνο για αιτήματα που έχουν φτάσει στον εξυπηρετητή με τη μέθοδο GET.

```
app.get("/path1", myMiddleware2)
app.get("/path2", myMiddleware1)
app.get(myMiddleware3)
```

Το Express.js διαθέτει συναρτήσεις για όλες τις διαθέσιμες μεθόδους HTTP, αν και οι get(), post(), put(), delete() είναι ίσως οι πιο συχνά χρησιμοποιούμενες. Γενικά λοιπόν, ο πιο απλός τρόπος δρομολόγησης είναι απευθείας στο αντικείμενο του Express.js:

```
app.METHOD(path, callback)
```

όπου

1. METHOD και path είναι η διαδρομή του αιτήματος
    1. METHOD είναι μια από τις μεθόδους HTTP (π.χ. GET, POST κ.λπ.),
    2. path είναι το μονοπάτι για το οποίο θα ισχύει ο δρομολογητής,
2. callback είναι η συνάρτηση που θα εκτελεστεί για τον συνδυασμό METHOD+path. Η συνάρτηση είναι πρακτικά η αρχή της αλυσίδας επεξεργασίας για την εκάστοτε διαδρομή METHOD+path.

12.3.6 Δρομολόγηση με τη μέθοδο app.route()

Δρομολόγηση με το αντικείμενο εφαρμογής του Express.js μπορεί να γίνει και με τη συνάρτηση `route()` του αντικείμενου εφαρμογής. Με τον τρόπο αυτό μπορούμε να ορίσουμε τον χειρισμό σε ένα συγκεκριμένο

μονοπάτι με ευανάγνωστο τρόπο, καθώς με τη route() ορίζουμε πρώτα το μονοπάτι και έπειτα όσες μεθόδους HTTP θέλουμε να προδιαγράψουμε για αυτό:

```
app.route('/recipe')
    .all((req, res) => {
        // το 'all' εκτελείται σε κάθε περίπτωση
    })
    .get((req, res) => {
        res.send('Επιστρέφει μια συνταγή')
    })
    .post((req, res) => {
        res.send('Προσθήκη της συνταγής')
    })
    .put((req, res) => {
        res.send('Ενημέρωση της συνταγής')
    })
```

12.3.7 Δρομολόγηση με την κλάση Router

Η δρομολόγηση με το αντικείμενο Router λειτουργεί όπως και με το αντικείμενο εφαρμογής του Express.js. Η διαφορά είναι πως το αντικείμενο Router υποστηρίζει μόνο τον μηχανισμό middleware και δρομολόγησης, ενώ το αντικείμενο εφαρμογής παρέχει περισσότερες λειτουργίες. Η δρομολόγηση με το αντικείμενο Router βοηθά στην οργάνωση όταν η εφαρμογή γίνεται πιο σύνθετη.

```
import express  from 'express'
const app = express()
const router = express.Router()

router.get('/random', function (req, res) {
  res.send('Επιστρέφει μια συνταγή')
})

router.get('/about', function (req, res) {
  res.send('Σχετικά με την εφαρμογή συνταγών')
})

app.use('/recipe', router)
```

Όπως βλέπουμε, η διαδρομή ορίζεται σε δύο βήματα. Πρώτα ορίζουμε τον δρομολογητή (router) και στη συνέχεια τον προσαρτάμε κάτω από ένα μονοπάτι. Στο τέλος, το παράδειγμά μας θα επιστρέφει μια τυχαία συνταγή σε αιτήματα που φτάνουν στο μονοπάτι “/recipe/random” και διάφορες πληροφορίες στο “/recipe/about”.

Σημειώστε πως θεωρούνται ίδιες οι διαδρομές

1. με ή χωρίς τελικό '/', π.χ. αίτημα στο /recipe είναι το ίδιο με /recipe/,
2. με μικρά ή κεφαλαία, π.χ. /recipe το ίδιο με /Recipe.

Η συμπεριφορά αυτή αλλάζει με τις αντίστοιχες ιδιότητες του αντικείμενου ρύθμισης του Router, strict και caseSensitive:

```
const options = {
    caseSensitive: true,
    strict: true
}
const router = express.Router(options)
```

12.4 Τα αντικείμενα αιτήματος και απόκρισης

Έχουμε δει μέχρι τώρα πώς να δρομολογήσουμε ένα αίτημα στην κατάλληλη αλυσίδα επεξεργασίας. Για να μπορέσουμε να επεξεργαστούμε κατάλληλα το αίτημα στα υποτμήματα της αλυσίδας επεξεργασίας,

χρειαζόμαστε πληροφορίες σχετικά με αυτό, που μας τις δίνει το αντικείμενο αιτήματος (request). Αντίστοιχα, μπορούμε να χρησιμοποιήσουμε το αντικείμενο απόκρισης (response) για να προδιαγράψουμε την απόκρισή μας στο αίτημα του πελάτη.

12.4.1 Το αντικείμενο αιτήματος

Το API του αντικείμενου Request μάς δίνει πρόσβαση στα δεδομένα του αιτήματος (Request API). Ίσως η πιο χρήσιμη ιδιότητα του αντικείμενου είναι η **request.query**, ένας πίνακας που περιέχει τις παραμέτρους που έστειλε ο πελάτης στο αίτημά του μέσω του query string.

Μπορούμε να δούμε τη βασική χρήση με ένα απλό παράδειγμα για την υποθετική εφαρμογή μας TastyRecipes. Έστω πως θέλουμε η εφαρμογή να επιστρέφει συνταγές με βάση χαρακτηριστικά που προσδιορίζει ο χρήστης στο αίτημά του, όπως π.χ. δυσκολία της συνταγής ή βαθμολογία:

```
import express from 'express'
const app = express()

app.get('/recipe/', function (req, res, next) {
    for (const key in req.query) {
        console.log(key, ":", req.query[key])
        next()
    }
    res.send("got a recipe.")
})

app.listen(3000, () => console.log('Ready'))
```

Ένα αίτημα όπως το http://localhost:3000/recipe?difficulty=easy&rating=5 θα τυπώσει στο τερματικό του εξυπηρετητή τα ονόματα των παραμέτρων (key) καθώς και την τιμή της καθεμίας (res.query[key]):

```
Ready
difficulty : easy
rating : 5
```

12.4.2 Ονοματισμένες παράμετροι

Ένας εναλλακτικός τρόπος να περάσουμε παραμέτρους με το URL του αιτήματος είναι να χρησιμοποιήσουμε τον μηχανισμό των ονοματισμένων παραμέτρων (named parameters). Οι τιμές των παραμέτρων θα είναι διαθέσιμες στο αντικείμενο request.parameters. Μπορούμε να ξαναγράψουμε το πιο πάνω παράδειγμα ώστε να αποκρίνεται σε αιτήματα σε URL της μορφής /recipe/<difficulty>/<rating>:

```
app.get('/recipe/difficulty/:difficulty/rating/:rating', function (req,
res, next) {
    console.log(req.params)
    res.send("got a recipe.")
})
```

Η κλήση στο URL http://localhost:3000/recipe/difficulty/easy/rating/5 θα έχει σαν αποτέλεσμα:

```
{ difficulty: 'easy', rating: '5' }
```

12.4.3 Πρόσβαση στα δεδομένα POST

Αιτήματα που έχουν φτάσει με τη μέθοδο POST, όμως, δεν μεταφέρουν την πληροφορία με το query string, αλλά περιέχεται στο σώμα του αιτήματος. Για να έχουμε πρόσβαση στα δεδομένα που έχουν σταλεί με τη μέθοδο POST, π.χ. από την υποβολή μιας φόρμας, χρειάζεται να χρησιμοποιήσουμε το middleware urlencoded, που είναι ενσωματωμένο στη βασική εγκατάσταση του Express.js:

```
app.use(express.urlencoded({ extended: true }))
```

```
app.post('/', (req, res) => {
    console.log("request body:", req.body)
    res.send("post ok")
})
```

12.4.4 Το αντικείμενο απόκρισης

Όταν τελειώσει η επεξεργασία του αιτήματος, κάποιο από τα υποτμήματα της αλυσίδας επεξεργασίας θα πρέπει να στείλει την απόκριση στον πελάτη. Η αποστολή της απόκρισης μπορεί να γίνει με διάφορες συναρτήσεις του αντικείμενου απόκρισης (response).

1. Response.send(message) Υπολογίζει το μέγεθος της απάντησης (Content-Length), τη στέλνει και σταματά την αλυσίδα επεξεργασίας.
2. Response.end() Στέλνει κενή απάντηση, χωρίς δεδομένα, και σταματά την αλυσίδα επεξεργασίας.
3. Response.status() Ορίζει την κεφαλίδα κατάστασης της απάντησης, π.χ. res.status(404).end()

Για να απαντήσουμε με κώδικα HTML, ο πιο απλός αλλά και κουραστικός τρόπος είναι να στείλουμε τον κώδικά μας με τη Response.send(), π.χ.:

```
app.get('/', (req, res) => {
    res.send('<html><head><meta charset="utf-8">.....')
})
```

Η πρακτική αυτή όμως είναι εμφανώς δύσχρηστη και για αυτό τον λόγο χρησιμοποιούμε μηχανές template (Ενότητα 12.5) για την προετοιμασία και αποστολή της απόκρισής μας, όπως θα δούμε στη συνέχεια. Για να αποστείλουμε την απάντησή μας μέσω μιας μηχανής template χρησιμοποιούμε τη συνάρτηση Response.render().

Τέλος, πολύ χρήσιμη είναι η δυνατότητα να αποστείλουμε απάντηση σε μορφή JSON, π.χ αν χρησιμοποιούμε το Express.js για να υλοποιήσουμε ένα REST API που θα είναι προσβάσιμο όχι από χρήστες αλλά από άλλες εφαρμογές. Για να στείλουμε την απάντηση σε μορφή JSON χρησιμοποιούμε τη συνάρτηση res.json(message)

12.5 Μηχανές template

Στα απλοϊκά παραδείγματα που έχουμε δει μέχρι τώρα η απόκριση της εφαρμογής μας στον χρήστη γίνεται με τη συνάρτηση Response.send(). Με τις μηχανές για template έχουμε απλές γλώσσες για τη δυναμική δημιουργία HTML.

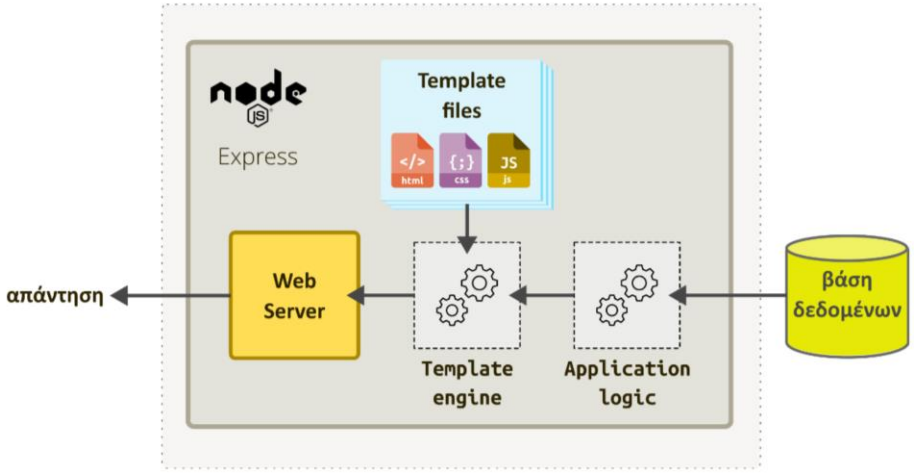
Μια μηχανή template συνδυάζει κάποια δεδομένα JavaScript ή JSON με κάποιο template για να παραγάγει HTML, το οποίο και τελικά θα επιστραφεί στον πελάτη.

Η επίσημη μηχανή για templates στο Express.js είναι η Pug. Αξίζει να σημειωθεί πως το παλιό της όνομα είναι Jade, και υπάρχουν ακόμη αναφορές σε αυτό το όνομα. Μια από τις πιο δημοφιλείς μηχανές, όμως, είναι η Handlebars.js, και αυτή είναι που θα δούμε στη συνέχεια καθώς είναι πιο απλή.

12.5.1 Βασική χρήση της Handlebars

Η χρήση μιας μηχανής template μάς επιτρέπει να διαχωρίζουμε τη λογική της εφαρμογής από την παρουσίαση της απόκρισης στον χρήστη. Η απόκρισή μας προετοιμάζεται στο κυρίως μέρος της εφαρμογής και, πριν σταλεί στον πελάτη, η μηχανή template αναλαμβάνει να την εμπλουτίσει με την κατάλληλη HTML.

Στην πραγματικότητα, η μηχανή template αναλαμβάνει να συνθέσει και να αποστείλει στον πελάτη μια σελίδα HTML που να περιέχει το αποτέλεσμα της επεξεργασίας του αιτήματός του (**Εικόνα *12.2***). Έχουμε δει στα προηγούμενα κεφάλαια πως η παρουσίαση πληροφορίας με τις τεχνολογίες HTML, CSS και JavaScript μπορεί γρήγορα να γίνει σύνθετη υπόθεση. Η μηχανή template Handlebars και γενικότερα όλες οι μηχανές template, σε διαφορετικό βαθμό η καθεμία, παρέχουν αρκετά εργαλεία ώστε ο κώδικας που συνθέτει την απόκρισή μας, εκτός από αποτελεσματικός, να είναι σπονδυλωτός και να μπορεί να συντηρηθεί και να επεκταθεί εύκολα. Τέτοια εργαλεία περιλαμβάνουν τη δυνατότητα να χρησιμοποιήσουμε διαφορετικά layout, π.χ. ένα layout για σελίδες που θα προβάλλονται σε αυθεντικοποιημένους χρήστες, άλλο για τους απλούς επισκέπτες, άλλο ίσως για τον διαχειριστή της εφαρμογής κ.ο.κ., ή τη δυνατότητα να συνθέσουμε τη σελίδα μας από διάφορα επιμέρους αποσπάσματα template κ.λπ.



Εικόνα 12.2 *Η μηχανή template συνθέτει την τελική απάντηση πριν αποσταλεί στον πελάτη.*

Στη συνέχεια θα δούμε τις βασικές δυνατότητες που παρέχει η μηχανή Handlebars, μέσα από ένα υποθετικό πρότζεκτ.

Η βασική δομή των αρχείων του πρότζεκτ μας θα είναι:

```
|-- index.mjs
|-- views/
|   |-- home.hbs
|   |-- layouts/
|   |   |-- main.hbs
|   |-- partials/
|   |   |-- header.hbs
|   |   |-- footer.hbs
```

όπου

1. '/views' είναι η θέση των template,
2. '/layouts/main.hbs' το κύριο layout, μέσα στο οποίο προβάλλονται τα template,
3. '/partials/*' επιμέρους αποσπάσματα template, για να φορτώνονται από άλλα template.

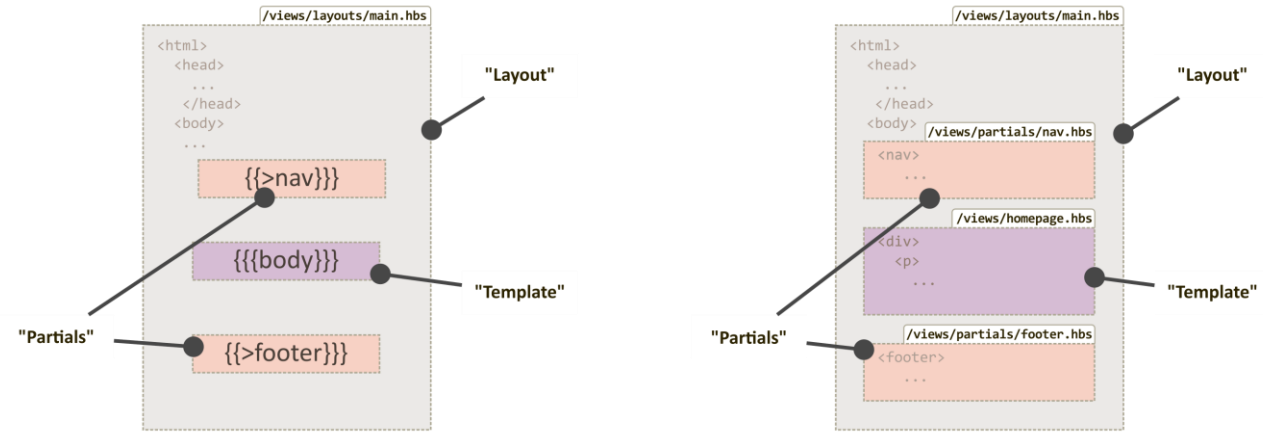
Για τη μηχανή Handlebars χρησιμοποιούμε το πακέτο express-handlebars (npm install express-handlebars). Στη βασική μας εφαρμογή (index.mjs) ορίζουμε ότι θα χρησιμοποιήσουμε τη Handlebars και ότι τα templates μας θα είναι σε αρχεία με κατάληξη '.hbs', αντί για τη προκαθορισμένη κατάληξη .handlebars:

```
import express from 'express';
import { engine } from 'express-handlebars';

const app = express()

app.engine('.hbs', engine({ extname: '.hbs' }));
app.set('view engine', '.hbs');
```

Η συνάρτηση `app.engine()` ορίζει ποια θα είναι η μηχανή template της εφαρμογής μας, δηλαδή εκείνη που θα καλείται με τη μέθοδο `res.render()` όταν θέλουμε να στείλουμε την απάντησή μας. Η συνάρτηση `app.set()` είναι μια βοηθητική συνάρτηση του Express.js με την οποία μπορούμε να προσδιορίσουμε την τιμή διαφόρων παραμέτρων, στην προκειμένη περίπτωση την κατάληξη των αρχείων της μηχανής template που χρησιμοποιούμε.



Εικόνα 12.3 *Στην ορολογία του Handlebars ένα template «τοποθετείται» μέσα σε ένα "view". Το template μπορεί να συνοδεύεται από επιμέρους "partial" τμήματα.*

Το template μας μπορεί να χωριστεί σε πολλά τμήματα. Το default layout '/views/layouts/main.hbs', περιέχει το βασικό layout μέσα στο οποίο θα προβληθεί το template, δηλαδή τον κώδικα HTML που θα πλαισιώνει όλες τις σελίδες της εφαρμογής μας (**Εικόνα *12.3***):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Tasty Recipes</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

Το εκάστοτε template θα προβληθεί στη θέση {{{body}}}. Για παράδειγμα, το template '/views/home.hbs' μπορεί να είναι πολύ απλό:

```
<main><p> Οι νόστιμες συνταγές: {{greeting}} </p></main>
```

Αλλά συνήθως χρειάζεται να μεταφέρουμε κάποια δεδομένα που θέλουμε να προβάλει η μηχανή template στην HTML. Μπορούμε να εμφανίσουμε τιμές μεταβλητών απλά ονοματίζοντάς τες σε διπλά άγκιστρα, όπως φαίνεται πιο πάνω, και να περάσουμε τις τιμές σαν παράμετρο στη συνάρτηση res.render():

```
app.get('/', (req, res) => {
  res.render('home', { greeting: 'Καλώς ήλθατε!'});
});
```

Η res.render() λοιπόν θα χρησιμοποιήσει το template './views/home.hbs', καθώς το πρώτο όρισμά της είναι το home. Επίσης, θα περάσει στο template αυτό το αντικείμενο του δεύτερου ορίσματος, το { greeting: 'Καλώς ήλθατε!'}. Το αποτέλεσμα θα προστεθεί στο default layout (που είναι το './views/layouts/main.hbs') και θα επιστραφεί τελικά στον πελάτη.

Σημειώστε εδώ πως το express-handlebars έχει προκαθορισμένη συμπεριφορά να αναζητά τα template στον φάκελο με όνομα /views και να χρησιμοποιεί το layout /views/layouts/main.hbs για να προβάλλει όλα τα άλλα template, όπως π.χ. το home.hbs. Δεν χρειάστηκε λοιπόν να τα προσδιορίσουμε αυτά στον κώδικά μας, αν και μπορούμε αν θέλουμε να αλλάξουμε αυτή τη συμπεριφορά.

Εκτός από το main.hbs, στον φάκελο './views/layouts/' μπορούμε να έχουμε και άλλα layout, που να αποτελούν τον βασικό σκελετό κάποιων από τις σελίδες μας:

```
|-- views
|   |-- layouts
|   |   |-- main.hbs
|   |   |-- other-layout.hbs
```

Για παράδειγμα, στον παρακάτω κώδικα το αίτημα στη διαδρομή /other-layout θα απαντηθεί κάνοντας render το template './views/home.hbs' στο layout './views/layouts/other-layout.hbs':

```
app.get('/other-layout', (req, res) => {
    res.render('home', { layout: "other-layout" })
})
```

Partials

Γρήγορα όμως θα χρειαστεί σε ένα πρότζεκτ να κατασκευάσουμε σύνθετες σελίδες, που αποτελούνται από επιμέρους template. Αυτό είναι πολύ συχνό, για παράδειγμα, σε ιστότοπους που το υποσέλιδό τους είναι πάντα το ίδιο, αυτό που συχνά ονομάζεται footer. Αντίστοιχα χρήσιμο είναι να έχουμε σε ξεχωριστό αρχείο τη γραμμή πλοήγησης ή την κεφαλίδα (header) με τον τίτλο και το λογότυπο του ιστότοπού μας.

Ο μηχανισμός των partials επιτρέπει να φορτώσουμε templates μέσα από άλλα templates. Στη δομή αρχείων του παραδείγματός μας έχουμε ορίσει το partial '/views/partials/footer.hbs', που μπορεί να περιέχει το εξής:

```
<footer><p> Το υποσέλιδο </p></footer>
```

Με αυτό τον τρόπο μπορούμε να επαναχρησιμοποιήσουμε τον κώδικα του footer.hbs σε πολλά διαφορετικά layout. Στο παράδειγμά μας έχουμε ήδη δύο layout, το main.hbs και το other-layout.hbs. Για να φορτώσουμε το partial στο layout μας, π.χ. στο main.hbs, γράφουμε:

```
...
{{> footer}}
</body>
```

12.5.2 Το αντικείμενο res.locals

Πολύ βολικό είναι το αντικείμενο res.locals. Ό,τι καταχωρίσουμε σε αυτό το αντικείμενο θα είναι διαθέσιμο στη μηχανή template χωρίς άλλες ενέργειες. Για παράδειγμα:

```
app.get("/", (req, res, next) => {
    res.locals.greeting = "Καλημέρα";
    next()
    }, (req, res) => {
        // δεν χρειάζεται να περάσουμε ρητά τη μεταβλητή "greeting"
        res.render('home')
    })
```

Τώρα στο template “home” μπορούμε απλά να γράψουμε

```
<main><p> Οι νόστιμες συνταγές: {{greeting}} </p></main>
```

Στατικά αρχεία

Πώς όμως θα διαθέσουμε στους πελάτες του εξυπηρετητή αρχεία που δεν χρειάζονται επεξεργασία; Τέτοια αρχεία είναι, για παράδειγμα, εικόνες, τα αρχεία CSS ή αρχεία με JavaScript που εκτελείται στον φυλλομετρητή. Αυτά ονομάζονται **στατικά αρχεία** και για να τα διαθέσουμε χρησιμοποιούμε άλλο ένα από τα ενσωματωμένα middleware του Express.js.

Το middleware static είναι αυτό που μας δίνει πρόσβαση σε στατικά αρχεία. Για να το χρησιμοποιήσουμε ορίζουμε έναν φάκελο που περιέχει στατικά αρχεία (ή και περισσότερους):

```
// Τα στατικά αρχεία μας θα βρίσκονται στον φάκελο /public
app.use(express.static('public'))
```

Όλα τα αρχεία που βρίσκονται στον φάκελο “public” θα είναι προσβάσιμα απευθείας, π.χ.

```
|-- public
|   |-- images
|       |-- mypic.png
|   |-- css
|       |-- mystyle.css
...
```

Τα αρχεία της παραπάνω δομής είναι προσβάσιμα από URL της μορφής:

1. http://localhost:3000/images/mypic.png
2. http://localhost:3000/images/mystyle.css

Προσέξτε πως στο URL έχουμε παραλείψει το μονοπάτι όπου φιλοξενούνται τα αρχεία, δηλαδή το public/.

12.6 Βοηθητικά εργαλεία της Handlebars

Η μηχανή Handlebars παρέχει και κάποια βοηθήματα (helpers) που μας δίνουν κάποια ευελιξία στη συγγραφή των template, όπως π.χ.:

1. #if:

```
{{#if greeting}}
    {{greeting}}
{{else}}
    Αντίο
{{/if}}
```

1. #unless: το αντίθετο του #if
2. #each: απαρίθμηση των στοιχείων ενός πίνακα

```
{{#each people}}
    <li>{{this}}</li>
{{/each}}
```

12.7 Handlebars - Expressions

Οι εκφράσεις περικλείονται από διπλά άγκιστρα. Η έκφραση {{greeting}}, όπως είδαμε πιο πάνω, θα αντικατασταθεί με την τιμή της μεταβλητής greeting. Αν, αντί για μεταβλητές, έχουμε αντικείμενα, μπορούμε να έχουμε πρόσβαση στις ιδιότητές τους με dot-notation, π.χ. για το αντικείμενο address που θα μπορούσαμε να περάσουμε στο template:

```
res.render( 'home', {   address: {  street: "Ypatias 1", city: "Patra" }
});
```

θα έχουμε διαθέσιμες τις τιμές του αντικείμενου address στη Handlebars:

```
<p>{{address.street}} {{address.city}}</p>
```

και

```
{{#with address}}
<p> {{street}} {{city}}</p>
{{/with}}
```

ή

```
{{#each address}}
<p> {{@key}}: {{this}}</p>
{{/with}}
```

12.8 Ερωτήσεις

1. Το Express.js είναι
    1. μια εξελιγμένη έκδοση της Node.js.
    2. ένα πακέτο της Node.js.
    3. μια αρχιτεκτονική που μπορούμε να χρησιμοποιήσουμε για να γράψουμε καλά δομημένα την εφαρμογή μας.
    4. ένας εναλλακτικός τρόπος να γράψουμε ασύγχρονες εφαρμογές στη Node.js.
2. Στη 2η γραμμή του παρακάτω αποσπάσματος κώδικα

```
import express  from 'express'
const app = express()
```

    1. η μεταβλητή app είναι το αντικείμενο εφαρμογής (application object) του Express.js.
    2. θα προκληθεί σφάλμα, καθώς η αρχικοποίηση ενός νέου στιγμιότυπου απαιτεί τη λέξη-κλειδί new.
    3. θα προκληθεί σφάλμα, καθώς δεν επιτρέπεται να χρησιμοποιηθεί η λέξη-κλειδί const με αυτό τον τρόπο.
    4. θα έπρεπε να γράψουμε const app = express.express()
3. Σε έναν χειριστή αιτήματος στο Express.js η εντολή res.send()
    1. είναι ισοδύναμη με την ακολουθία στη Node.js: res.write();res.end()
    2. θα προκαλέσει σφάλμα.
    3. είναι ισοδύναμη με την ακολουθία στη Node.js: res.writeHead();res.end()
4. Ένα υποτμήμα της αλυσίδας επεξεργασίας είναι
    1. μια συνάρτηση με ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
    2. μια κλάση τύπου Express, που περιέχει μια συνάρτηση με ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
    3. μια συνάρτηση που είναι μέλος αντικείμενου εφαρμογής (application object) και που έχει ορίσματα τα αντικείμενα αιτήματος και απόκρισης.
5. Επιλέξετε όσα ισχύουν: Ένα υποτμήμα της αλυσίδας επεξεργασίας τερματίζει τη λειτουργία του με τους εξής τρόπους:
    1. Προωθεί το request και το response object στο επόμενο υποτμήμα με τη next()
    2. Επιστρέφει τον έλεγχο ροής στο αντικείμενο εφαρμογής του Express.js με τη return.
    3. Αποκρίνεται στο αίτημα του πελάτη.
    4. Επιστρέφει τον έλεγχο ροής στο αντικείμενο εφαρμογής του Express.js με κλήση σε συνάρτηση επιστροφής.
6. Μια διαδρομή (route) καθορίζεται από:
    1. τη μέθοδο HTTP (HTTP method).
    2. το μονοπάτι του αιτήματος (path).
    3. τη θύρα του εξυπηρετητή (server port).
    4. την αλυσίδα επεξεργασίας που θα χρησιμοποιηθεί.
    5. τον μηχανισμό βιβλιοθηκών που χρησιμοποιούμε (commonjs/ES6).
7. Έστω η παρακάτω αλυσίδα επεξεργασίας

```
app.get("/path1", MW1)
app.use(MW2)
app.get("/", MW3)
```

    Επιλέξετε όσα ισχύουν:
    1. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW2->MW3
    2. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW2
    3. με αίτημα στο μονοπάτι “/path1”, η αλυσίδα επεξεργασίας είναι MW1->MW3
    4. με αίτημα στο μονοπάτι “/”, η αλυσίδα επεξεργασίας είναι MW3
    5. με αίτημα στο μονοπάτι “/”, η αλυσίδα επεξεργασίας είναι MW2->MW3
8. Επιλέξτε όσα από τα παρακάτω αποτελούν έγκυρους τρόπους να δηλωθεί η αλυσίδα επεξεργασίας:
    1. app.use(path, callback)
    2. app.get(path, callback)

3. app.get(callback).post(callback)
4. app.route(path).all(callback)
5. app.route(path).get(callback).post(callback)

9. Η ιδιότητα request.query του αντικείμενου του αιτήματος HTTP περιέχει της παραμέτρους του αιτήματος HTTP όταν αυτό γίνεται με τη μέθοδο GET είτε με την POST.
   – Σωστό/Λάθος
10. Η ιδιότητα request.parameters του αντικείμενου του αιτήματος HTTP περιέχει τις παραμέτρους του αιτήματος HTTP όταν αυτό γίνεται με τη μέθοδο GET είτε με την POST.
   – Σωστό/Λάθος
11. Με ποιον από τους παρακάτω τρόπους μπορούμε να αποκτήσουμε πρόσβαση στα δεδομένα του αιτήματος στο παρακάτω τμήμα;

```
app.get('/name/:name/age:age', function (req, res, next) {
    --> Επιλέξτε τι θα μπει εδώ για να τυπωθούν οι παράμετροι και οι τιμες
τους <--
    res.send("ok")
})
```

1. console.log(req.params)
2. console.log(req.body)
3. console.log(req.query)

12. Με τη response.status(404) ορίζουμε την κεφαλίδα απόκρισης και τη στέλνουμε στον πελάτη.
   – Σωστό/Λάθος
13. Επιλέξετε με ποιους από τους παρακάτω τρόπους μπορούμε να στείλουμε μια απάντηση στον πελάτη και να τερματίσουμε την αλυσίδα επεξεργασίας.
   1. res.send()
   2. res.end()
   3. res.status()
   4. res.render()
   5. res.json()
14. Το αντικείμενο **Router** χρησιμοποιείται όπως και οποιοδήποτε middleware.
   – Σωστό/Λάθος
15. Όταν η εφαρμογή είναι σύνθετη, προτιμάμε την οργάνωση των διαδρομών με (επιλέξτε το σωστό)
   1. app.METHOD(path, callback, όπου method οποιοδήποτε από τα ρήματα του HTTP (get, post κ.λπ.).
   2. app.route(path).METHOD(callback).
   3. app.route(path, router), όπου router ένα αντικείμενο τύπου Router.
16. Η προκαθορισμένη συμπεριφορά στη δρομολόγηση με το Express.js είναι:
   1. Όπως και στη Node.js.
   2. Έχει σημασία αν τα γράμματα του μονοπατιού είναι πεζά η κεφαλαία, αλλά δεν μας ενδιαφέρει αν υπάρχει ή όχι τελικό "/".
   3. Έχει σημασία αν υπάρχει ή όχι τελικό "/", αλλά δεν διακρίνουμε μικρά ή κεφαλαία γράμματα.
   4. Δεν έχει σημασία το τελικό "/", ούτε γίνεται διάκριση πεζών και κεφαλαίων γραμμάτων.
17. Όταν λέμε στατικά αρχεία εννοούμε αρχεία τα οποία το Express.js επιστρέφει όπως είναι, χωρίς επεξεργασία από κάποιο ειδικό υποτμήμα.
   – Σωστό/Λάθος
18. Σε μια εφαρμογή Express.js μπορούμε να ορίσουμε μόνο έναν φάκελο που θα περιέχει τα στατικά μας αρχεία.
   – Σωστό/Λάθος
19. Τα στατικά μας αρχεία μπορούν να βρίσκονται μόνο σε φάκελο με όνομα static ή public.
   – Σωστό/Λάθος
20. Έστω η εφαρμογή μας "http://localhost/" και ότι έχουμε γράψει:

```
import express  from 'express'
const app = express()
app.use(express.static('/public'))
```

Τότε, για να έχουμε πρόσβαση στο αρχείο css, που βρίσκεται π.χ. στο /public/mystyle.css, θα κάνουμε αίτημα στο

1. http://localhost/static/public/mystyle.css
2. http://localhost/public/mystyle.css
3. http://localhost/mystyle.css

21. Έστω η εφαρμογή μας “http://localhost/” και ότι έχουμε γράψει:

```
import express  from 'express'
const app = express()
app.use(express.static('/public'))
```

Τότε, για να φορτώσουμε από την HTML το αρχείο css, που βρίσκεται π.χ. στο /public/mystyle.css, θα γράψουμε στην περιοχή <head>

1. <link rel="stylesheet" href="/static/public/mystyle.css">
2. <link rel="stylesheet" href="/public/mystyle.css">
3. <link rel="stylesheet" href="/mystyle.css">

22. Με την παρακάτω εντολή μπορούμε να χρησιμοποιήσουμε τη μηχανή template για να κατασκευάσει και να στείλει την απάντησή μας στον πελάτη:
    1. res.engine()
    2. res.send()
    3. res.render()
    4. res.end()
    5. res.template()
23. Για να προσδιορίσουμε ποια μηχανή template,
    1. καλούμε τη μέθοδο app.enable()
    2. καλούμε τη μέθοδο app.set()
    3. καλούμε τη μέθοδο app.engine()
    4. δεν χρειάζεται να κάνουμε κάτι ιδιαίτερο, καθώς το Express.js υποστηρίζει εγγενώς τη χρήση μηχανών template.
24. Στο layout μας, για να υποδείξουμε το σημείο που θέλουμε να προβληθεί το template, χρησιμοποιούμε την παρακάτω σύνταξη:
    1. `{{{body}}}`
    2. `{{{template}}}`
    3. `{{{view}}}`
    4. `{{{hbs}}}`
25. Το προκαθορισμένο layout είναι αποθηκευμένο στο αρχείο
    1. '/views/layouts/main.hbs'
    2. '/views/main.hbs'
    3. '/views/layout.hbs'
    4. '/views/templates/main.hbs'
26. Στο handlebars, το template μας περικλείεται από ένα layout.
    – Σωστό/Λάθος
27. Το layout μέσα στο οποίο περιέχεται το template πρέπει να είναι υποχρεωτικά αποθηκευμένο σε αρχείο με όνομα /views/layouts/main.hbs
    – Σωστό/Λάθος
28. Το βοήθημα `{{#if}}` θα επιστρέψει true αν
    1. στην έκφραση {{#if property value}} ισχύει property==value
    2. στην έκφραση {{#if property==value}} ισχύει property==value
    3. στην έκφραση {{#if property}} το property είναι διάφορο των false, undefined, null, 0 ή []
29. Το βοήθημα `{{#each}}` μπορεί να διατρέξει τα στοιχεία ενός πίνακα και να τυπώσει τη θέση, το όνομα και την τιμή κάθε στοιχείου με:
    1. `{{#each object}} {{@index}}: {{@key}}: {{this}}  {{/each}}`
    2. `{{#each object}} {{index}}:  {{key}}:  {{value}}  {{/each}}`
    3. `{{#each object}} {{@index}}: {{@key}}: {{@value}} {{/each}}`

12.9 Βιβλιογραφία και Αναφορές

Όπως και στο κεφάλαιο για τον προγραμματισμό στο περιβάλλον της Node.js (κεφ. 11), έτσι και εδώ το περιεχόμενο αφορά την παρουσίαση και χρήση μιας τεχνολογίας. Συνεπώς η κύρια πηγή πληροφοριών και η πρώτη στάση για περαιτέρω εμβάθυνση στο θέμα είναι και εδώ η επίσημη ιστοσελίδα του Express.js. Εκεί μπορεί κανείς να βρει την αναλυτική τεκμηρίωση του API, οδηγούς για χρήστες που αποκτούν την πρώτη επαφή, αλλά και άρθρα που αφορούν εξειδικευμένα θέματα και που ενδιαφέρουν τους προχωρημένους χρήστες.

Η κύρια πηγή και για τη μηχανή Handlebars είναι ο οδηγός που βρίσκεται στην επίσημη ιστοσελίδα. Όπως και για το Express.js, ο οδηγός συνοδεύεται και από την τεκμηρίωση της βιβλιοθήκης της Handlebars.

A. Ξενόγλωσσες

Συλλογικό (2022). *Express.js4.x API Reference*. Express.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://expressjs.com/en/4x/api.html

Συλλογικό (2022). *Node.js API Reference Documentation*. Node.js. Ανακτήθηκε στις 14 Σεπτεμβρίου 2022 από https://nodejs.org/en/docs/

B. Ελληνόγλωσσες

Σιντόρης, Χ., & Αβούρης, Ν. (2022). Ανάπτυξη διαδικτυακών εφαρμογών με Node.js. Ανοικτό διαδικτυακό μάθημα, https://mathesis.cup.gr.