# Express introduction

Our first server application

# Our goals

- What is express and why do we need it?
- Understand Express architecture
- Understand the patterns we use with express
- Using those patterns solve common server problems like
  - request body
  - session
  - Authentication / Authorization
- We will also learn how to split our server files and arrange them properly for a well structured application

# What is Express

- Express is a framework for creating web server
- It will get a request and send a proper response
- Minimalistic
- Unopinionated
- Fast
- Very easy learning curve
- Large community that develops a lot of plugins

# Express architecture - App - EX

- Building an express app starts with creating an express app
- An app contains methods to run on each request
- Usually we will have one app
- The app listens to a certain port for requests
- Let’s start by installing express and starting to listen on a port
- The app will listen for request and activate certain methods on certain requests

# Express architecture - Middleware - EX

- A middleware is a function that will be called on certain requests
- That function have access to the request, response, next
  - request - object containing information about the current request
  - response - used to send the response back
  - next - used to pass to the next middleware
- Add your first middleware, on all get requests, send a response of hello world

# Express architecture - Attaching a middleware

- Attaching a middleware is used with app.[METHOD] where method can be one of the following:
- The middleware will work based on the request type
  - use
    - used for added behaviour and usually not to deal with routes usually placed at the top
    - optional path and will match path prefix as well
  - all
  - get
  - post
  - put
  - delete
  - patch
  - options
  - head

# Express architecture - middleware path

- The first argument of the middleware is the path
- With use the default value of the path is “/” which means it will work on all the requests
- The match on the not use route methods should match exactly
- The path can be:  regular expression or string, or array of strings / regex
- path can transfer params in the route (query or fragment do not effect the match)
- Express use a library called path-to-regexp
  - https://www.npmjs.com/package/path-to-regexp
- Let’s examine some path examples

# Express architecture - middleware path

- app.use(‘/user’, …)
  - will match /user  or /user/foo/bar/hello/wold
  - Will not match / or /userhello
- app.all(‘/user’, …)
  - Will match /user
  - Will not match /user/foo/bar/…

- app.get(‘/user/:id’)
  - will match /user/30
  - req.params.id === ’30’
- app.get(‘/user/:id(\\d+)’, …)
  - will match /user/30 and req.params.id === ’30’
  - will not match /user/foo

# Express architecture - middleware path

- app.get(/\/user/)
  - will match /user or /sdfasdf/user/sdfsdf or /sdf/userfasdf
- app.all('/user/*', …)
  - will match /user/ or /user/foo/bar
  - will not match /user

# Express architecture - EX

- Create a server that will get a number param on the url and print that param

# Express architecture - middleware function

- The middleware function has the following signature
  - function(req, res, next)
- req is the request object
- res represents the response
- next will pass to the next middleware
- your choice is to either call next or to return the response
- when passing to next we can change the request response object
- You can use async functions as well
- You can place one function, list of functions, array of function

# Express architecture - middleware function - req

- An object representing the request
- common properties:
  - req.params
  - req.headers
  - req.query
  - req.url
  - req.method
- It is common to add key values to the response

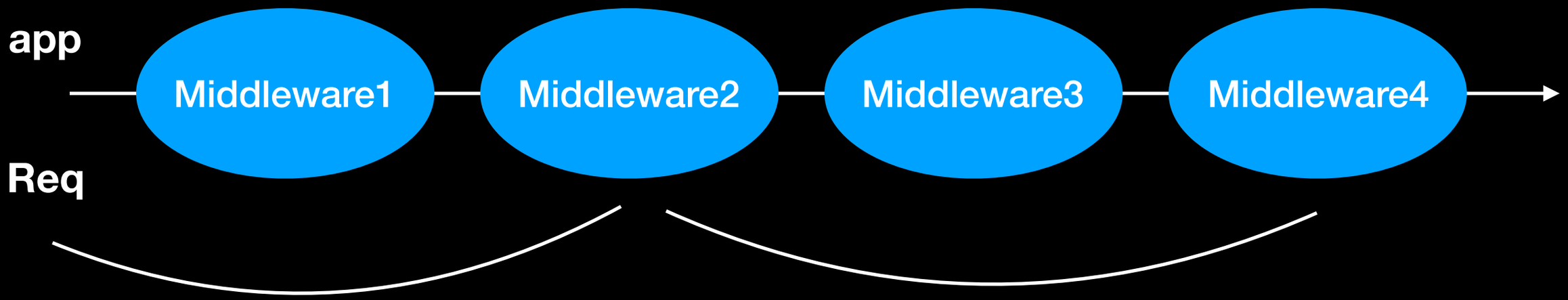
# Express architecture - middleware function - res

- An object representing the response
- Useful properties and methods
  - res.send
  - res.json
  - res.status
  - res.redirect
  - res.sendFile

# Express architecture - middleware function - next

- next is in charge to pass the req and response to the next middleware function in line
- the next middleware function can belong to the same app.[method] or a different one
- To skip the rest of the functions in this app.[method] and jump to the next app.[method] you can pass next('route')
  - will not work with use
- To pass an error we do next(new Error(…))

# Express architecture



# Express pattern - middleware creator -EX

- This pattern is used to send options to our middleware
- We use closure to place variable configuration values above the middleware function.
- Let’s exercise this pattern by creating a configurable middleware which we pass a name string to it
- That middleware should print ‘hello <name>’

# Express pattern - community middleware

- Express is a very popular node framework
- large community develops a lot of 3rd party middleware
- Almost all of those middleware we attach with the create middleware pattern we learned before
- Each middleware we have to supply it’s own configurations.
- Let’s go over the popular middlewares

# Express - community middleware - static

- express.static - The static middleware is used to serve static files
- We need to give it the path to the folder where our static files are located
- When requesting a static file we will not need to
- We can give create a virtual folder by providing the path.
- As an ex:
  - create a static assets directory with an image
  - install the static middleware
  - activate the server and make sure you can request the image and see it in the browser
- Do you think it’s a good idea to let express serve static files?

# Express pattern - adding to the request - EX

- Another pattern commonly used in express is modifying the request object
- The request object is passed between middlewares so it can be used to transfer data from middleware to middleware
- if we will add a key to the request object with data, the same request object with the same key and data is passed to the next middleware
- Let’s create the following middleware
  - Check if we have an Authorization header
  - if so create a req.user with the data that was passed in that header
  - Otherwise pass a status 401 with an unauthorized message

# Express pattern - adding to the request

- Let’s examine popular common community middlewares that use this pattern to help us solve problems

# Express pattern - adding to the request

- Let’s examine popular common community middlewares that use this pattern to help us solve problems

# Express pattern - request body - EX

- Some requests like post and put are sent with data from the user in the request body
- That data can be sent in different format : form-data, json, etc.
- We would like to grab the data sent in our express middleware functions
- body-parser is a middleware that will populate req.body with the data the user sent in the body of the request
- That data needs to be validated
- EX: add this middleware to parse json data sent in the request body

# Express pattern - Strategy pattern

- In the strategy pattern the different strategies need to implement a certain interface
- Our middleware can use different strategy to perform a certain action.
- In express this pattern usually look like this

```
app.use(middlewareCreator({
    strategy: new SomeCommunityStrategy(...)
}))

middlewareCreator.attachStrategy(new SomeStrategy(...));
app.use(middlewareCreator.initialize())
```

- Let’s examine some middlewares that use this pattern

# Express pattern - sessions

- The session information is usually stored on the server
- the server sends a cookie to the client with a hashed encrypted session id
- The client will send his cookies with the encrypted session id
- the server will decrypt the session id and use it to retrieve the user session data
- Where we save the session data can vary
  - Memory (dev only)
  - Database
  - Memory database (Redis, Memcached)
  - Cookies

# Express pattern - express-session middleware

- With express-session we can use sessions in our express app
- The session will be added to the request object
  - req.session
- req.session is a regular js object which you can retrieve keys and set key values
- express-session can store the session data in all the common available options we just need to set the appropriate store
- The default store is memory and is not recommended
- EX:
  - create 2 pages
  - the first one the user enters text in a text input and when he submits the form we save what he typed in the session
  - the second page displays that data from the session

# Express pattern - express-session strategy

- The express-session middleware uses the strategy pattern to decide how to store the session data
- By default it is stored in the memory
- EX:
  - Switch the strategy to store the data in the file system
  - The strategy to do that is called session-file-store
  - the option to specify a new storing strategy is called store

# Express pattern - Authentication

- Dealing with authentication in express is also done via the pattern of adding data to the request object combined with the strategy pattern
- After the user is authenticated we are adding the user property to the request
  - req.user
- We attach the strategy of how we want to authenticate where a strategy can be
  - username and password
  - facebook
  - twitter
  - etc.
- The middleware we want to use for authentication is called passport

# Express pattern - passport

- Passport is an authentication middleware
- You connect an authentication strategy to passport
- That strategy can be something that you write but usually you will use community strategies
- Passport will take care of placing req.user
- Passport will take care of persistent login session (needs to be disabled when not needed)
- To connect passport we need the following

# Express - passport - 1. Strategy

- Strategy determines the way we will authenticate the user
  - username + password
  - JWT
  - Facebook
- We attach it via passport.use(new OurStrategy(…))

# Express - passport - 2. Verify Callback

- The verify callback will get the authentication data and will need to
  - find the user that match the authentication data
  - if not finding a user return false
  - If error we can return an error
- For example for the username password strategy we can do the following
  - Look for a user with that username (if not return false)
  - found a user then verify that the password match and if so return the user
- The verify callback will get a done method as last argument and will need to call with error as first argument and the user as the second (or false if authentication failed)

# Express - passport - 3. Session?

- When adding our authentication we need to decide if we want to persist the user login with a session
  - Can you give example where we would like to persist with a session and an example where we wouldn’t?
- To persist to session we have to do the following
  - Add express-session middleware
  - Add passport.session() middleware after passport.initialize()
  - implement passport.serializeUser(function(user, done) { …})
  - implement passport.deserializeUser(function(id, done) { … })

# Express passport - EX

- For our first ex. we will use passport to authenticate with credentials
- the user will provide a username and a password and we will need to authenticate him
- We will create an object containing the usernames and password of our users (usually this will reside in the database)
- We will connect the proper strategy for the job which is called passport-local and requires a verify callback
- We will persist the login to the session
- passport-local also required bodyParser installed

# Express passport - Authentication in REST server

- For REST server it is common to authenticate using some sort of token authentication
- The user will enter his username and password and upon successful login he will get a token
- The token will have to be attached to any subsequent requests the user makes
  - The token is usually attach to the request as a header usually the header is called Authorization
- This authentication does not require sessions

# Express passport - JWT

- JWT is a popular token authentication standard
- After user logs in he will recieve a JWT token which he needs to pass in the Authorization header
  - Authorization: Bearer <token>
- The token is encrypted using a secret, only the one that holds the secret can decrypt the token (the server holds the secret)
- The token contains data the can only be viewed by the one who holds the secret
- Although it is protected we do not store sensitive information in the token data
- We can store information in the token like the user primary key that we can use to grab the user from the database
- Let’s examine the JWT structure: jwt.io

# Express passport - JWT - EX

- Let’s create a login page that will issue a token on success login
- the user can grab that token and send a request to a restricted route
- We will use Passport for authentication
- We will use passport-jwt strategy
- For the login page you can use the username and password strategy we used before (this time we do not require to use sessions)
- to create the token we will use the package node-jsonwebtoken
- We will add the user id in the payload of the token

# Express - Authorization

- The popular way to deal with authorization is by using the roles system
- A role is identified by a string name
- A resource is identified by a route with access to data (data is usually matching our database tables)
- Every resource has a set of actions that we can preform on the data
  - read, write, delete, update…
- Each role can perform certain actions on resources
- The roles are usually arranged by parent child hierarchy. for example
  - Admin - can do everything
  - User - can read all users and update only my user
  - Guest - can only login

# Express - Authorization Roles - EX

- From the previous ex, add a role to every user
- create a single user with admin role
- only the admin can view all the users
- all the other users will receive a 403 error

# Express - Authorization - ACL Authorization in database

- When the authorization get's a bit more complex we will need to save the roles in the database
- acl is a package that can connect to different databases and help us persist the roles in the database.
- It will also help us manage the roles
- create new roles
- create parent child roles

# Express pattern- Router

- You can use Router to split your server application to chunks
- each chunk you can use router.[METHOD] to attach a middleware
- Routers help us split our application to different files
- You connect the Router to the main app with app.use(router)
- It helps to look at the urls of your app to get inspired how to split your app to routers
- For example if you are creating a REST server with the following api's
  - /user
  - /user/:id
  - /task
  - /task/:id
- If those are your urls then consider having 2 Routers one for user and one for task

# Express pattern- Router - EX

- Create a new file user.js that will contain a user router
- The user router will define a route to grab list of users and define a route to grab a single user
- Connect that router to the main express app

# Express app configuration

- Express app configuration determines how our express app will behave
- When creating express app we have default configurations set for us
- We can change the configuration using:
  - app.set(key, value)
- We can get the value of a configuration:
  - app.get(key)
- Few configuration options we will examine
  - views - the directory of the application views
  - view engine - the view engine to use to render the views

# Express Views - Template engine

- We create the views using template engine
- Template engine convert a text syntax + data to html
- The text syntax will vary between each template engine
- Think about the middleware as our controller and the template engine as our view so our server application turns into a model view controller architecture
- express supports all the popular template engines
- We recommend using
  - 1. Pug
  - 2. Mustache
- In this lesson we will integrate Mustache with express which is html based and easier to learn
- Pug has more built in features but is not html based and has a bit harder learning curve

# Express Views - configuration views - EX

- the views configuration need to be set to the directory where you view files will reside
- the template files will end with mst extension
- EX:
  - create a folder called views where  our template files will reside
  - point the views configuration to that file.

# Express Views - set the view engine

- you connect a template engine using app.engine(ext, theEngine)
- you set the default engine to use using app.set('view engine', ext)
  - this will be used when the extension is omitted or different then what is specified
- Install mustache-express and set it as the template engine

# Express Views - Your first template - EX

- Create your first template index.mst with an hello message
- Try to pass data between the middleware and the template

# Express pattern - Error handling

- Express already has an error handler that will catch error on sync code
- On async code we need to transfer the error to the next handler
- By default express will not catch errors in the async code
- you can place error handler by attaching app.use with a function that contains 4 arguments, the first one is the error and the rest is re, res, next

# Express Generator - arranging the files

- The express generator helps you bootstrap an express project
- It will arrange the files in one of the recommended ways to arrange your project files
- It can connect a template engine
- the package express-generator is usually installed globaly
- Let’s try and install this package and bootstrap a new application

# Summary

- In this lesson we covered the patterns we use on an express application
- We took those patterns and solve common server problems like
  - Session
  - request body
  - Authentication
  - Authorization
- We also saw how we create view and attach a view engine to our app.