

Develop a Reliable Backend with Node and Express



Response Methods, Error Handling, and CRUD Operations in Express.js



Learning Objectives

By the end of this lesson, you will be able to:

- 🕒 Analyze different response methods to understand the varied ways of responding to requests made by users
- 🕒 Determine how to run an application, ensuring proper execution to cater to user requests
- 🕒 Illustrate the use of `app.locals`, `app.render()`, `app.routes`, and `app.listen()` method to utilize specific functionalities and configurations within the Express application
- 🕒 Demonstrate create, update, read, and delete operations, allowing for interaction and manipulation of information as needed for user-related functionalities



A Day in a Life of a MERN Stack Developer

Chester is working as a MERN stack developer for an organization and has been assigned a project. The project has multiple issues. For instance, the application throws multiple errors, and there is no fixed mechanism to manage them.

He analyzes the code and understands that error handling is something that needs to be focused on for this project.

In this lesson, you will be able to explore more about error handling and its methods to understand how it could be implemented in the above project.





Response Methods

Response Methods

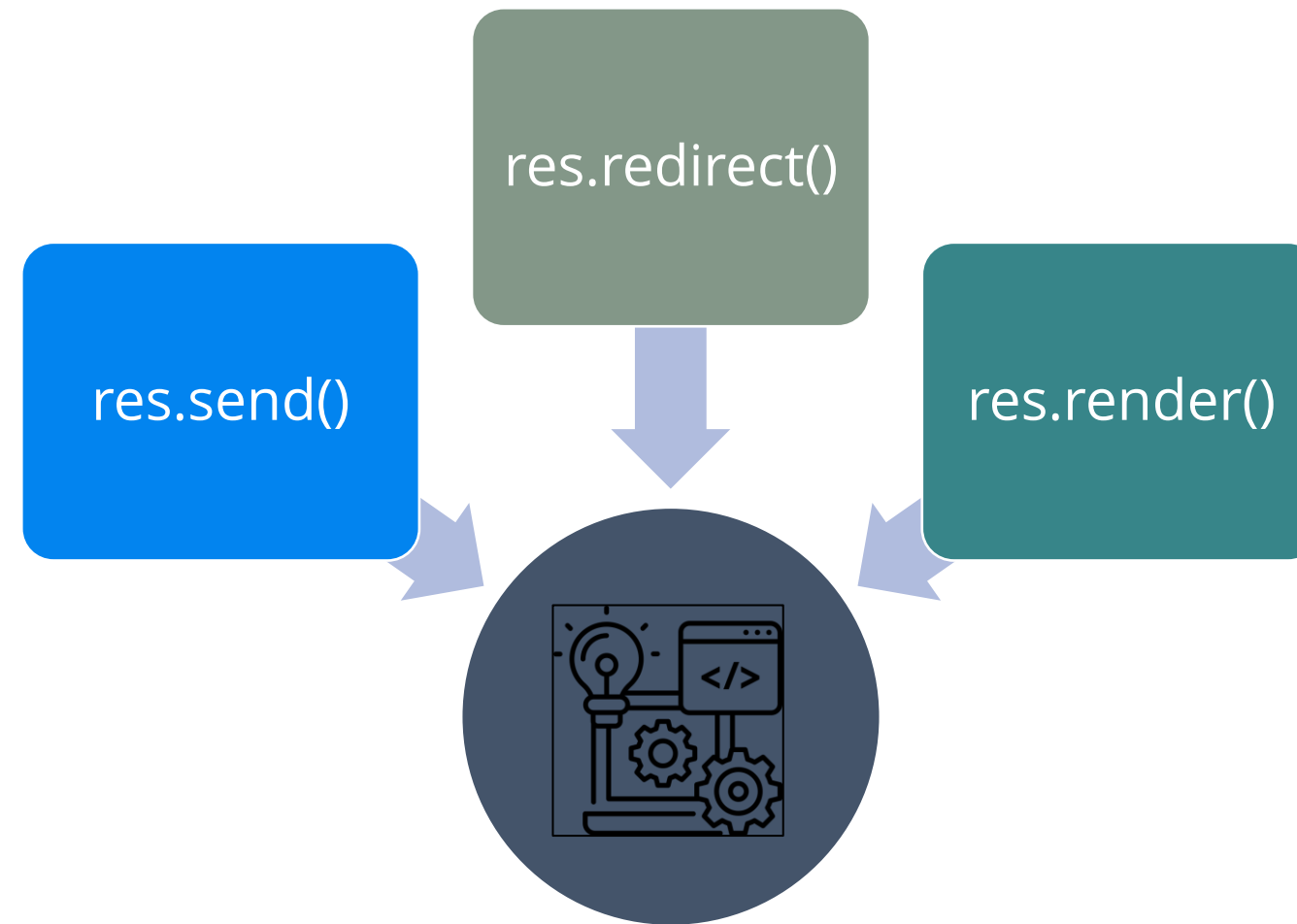
The Response object (res) specifies the HTTP response that an Express.js app sends when it receives an HTTP request.



It allows you to enter new cookie values and have them written to the client browser.

Response Methods

In response method, the following objects cannot be used again, or else an uncaught error will occur.



express.json()

The `express.json()` method is an Express built-in middleware function used for parsing incoming requests with the JSON payloads.

```
Syntax:  
express.json([options])
```

This method returns a middleware that only parses JSON and examines requests where the content-type header matches the type option.

Properties of express.json()

Following are the properties of the options object available with this method:

Property	Purpose
inflate	This enables or disables the deflated or compressed bodies' handling. True is the default value.
limit	This option controls the maximum size of the request body.
reviver	As the second argument, this option is passed to the JSON.parse method.
strict	This toggles the acceptance of arrays or objects.
type	This specifies the media type for the middleware to be parsed.

Example

Make a file called **express.js** and paste the following code into it. To run the following code, use the command **node express.js** after creating the file:

```
// Importing the express module
var express = require('express');

// Initializing the express and port number
var app = express();
var PORT = 3000;

// Calling the express.json() method for parsing
app.use(express.json());

// Reading content-type
app.post('/', function (req, res) {
  console.log(req.body.name)
  res.end();
})

// Listening to the port
app.listen(PORT, function(err){
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

Example

Before calling the API endpoint:, set these properties:

1. In headers, set the content-type to application or json.
2. In the POST request, include the following body - {name: Skillcart}

```
Output:  
C:\home  
ode>> node expressRaw.js  
Server listening on PORT 3000  
Skillcart
```

express.raw()

The `express.raw()` method is an Express built-in middleware function that is used for parsing incoming requests into a buffer.

```
Syntax:  
express.raw([options])
```

This method returns middleware that parses all JSON bodies as buffers and only considers requests where the content-type header matches the type option.

Properties

The following are the properties of the options object available with this method:

Property	Purpose
inflate	This enables or disables the deflated or compressed bodies' handling. True is the default value.
limit	This option controls the maximum request body size.
type	This option determines the media type that will be parsed for the middleware.

Example

Make a file called **expressRaw.js** and paste the following code into it. After creating the file, use the command **node expressRaw.js** to run this code.

```
// Importing the express module
var express = require('express');

// Initializing the express and port number
var app = express();
var PORT = 3000;

// Using the express.raw middleware
app.use(express.raw());

// Reading content-type
app.post('/', function (req, res) {
  console.log(req.body)
  res.end();
})

// Listening to the port
app.listen(PORT, function(err){
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

Example

Before calling the API endpoint:, set these properties:

1. In headers, set the content-type to application or octet-stream.
2. In the POST request, include the following body - {name: Skillcart}

```
Output:  
C:\home  
ode>> node expressRaw.js  
Server listening on PORT 3000  
Skillcart
```

express.Router()

Use the `express.Router()` function to create a new router object

```
Syntax:  
express.Router( [options] )
```

Use this function to create a new router object to handle requests

Properties

The following are the properties of the options object available with this method:

Property	Purpose
case-sensitive	This allows for case sensitivity
mergeParams	It keeps the parent router's req.params values.
strict	This allows for strict routing.
return value	The new router object is returned by this function.

express.Router()

Installing the express module:

```
Syntax:  
express.Router( [options] )
```

The above command can be used to install this package.

express.Router()

After installing the express module, use the command to check express version in command prompt.

```
npm version express
```

express.Router()

After that, create a folder and add a file, such as index.js. Type the following command to run this file:

```
node index.js
```

Example

Filename: index.js

```
var express = require('express');
var app = express();
var PORT = 3000;

// Single routing
var router = express.Router();




router.get('/', function (req, res, next) {
    console.log("Router Working");
    res.end();
})

app.use(router);

app.listen(PORT, function(err){
    if (err) console.log(err);
    console.log("Server listening on PORT", PORT);
});
```

Example

The project structure is given below:

Name	Date modified	Type	Size
 node_modules	06-06-2020 04:55 PM	File folder	
 index.js	06-06-2020 04:54 PM	JS File	0 KB
 package-lock.json	06-06-2020 04:55 PM	JSON File	14 KB

Output

Check that you have installed the express module by running the following command:

```
npm install express
```

Use the following command to run the index.js file:

```
node index.js
```


Output

Following it the output received for the previous code:

```
Output:  
Server listening on PORT 3000
```


Output

Now, open the browser and navigate to <http://localhost:3000/> where the following output is obtained:

A terminal window with a blue title bar and a light gray body. It contains the following text:

```
Output:  
Server listening on PORT 3000  
Router Working
```

```
Output:  
Server listening on PORT 3000  
Router Working
```

As seen in the above output, the Router is working properly.

`express.static()`

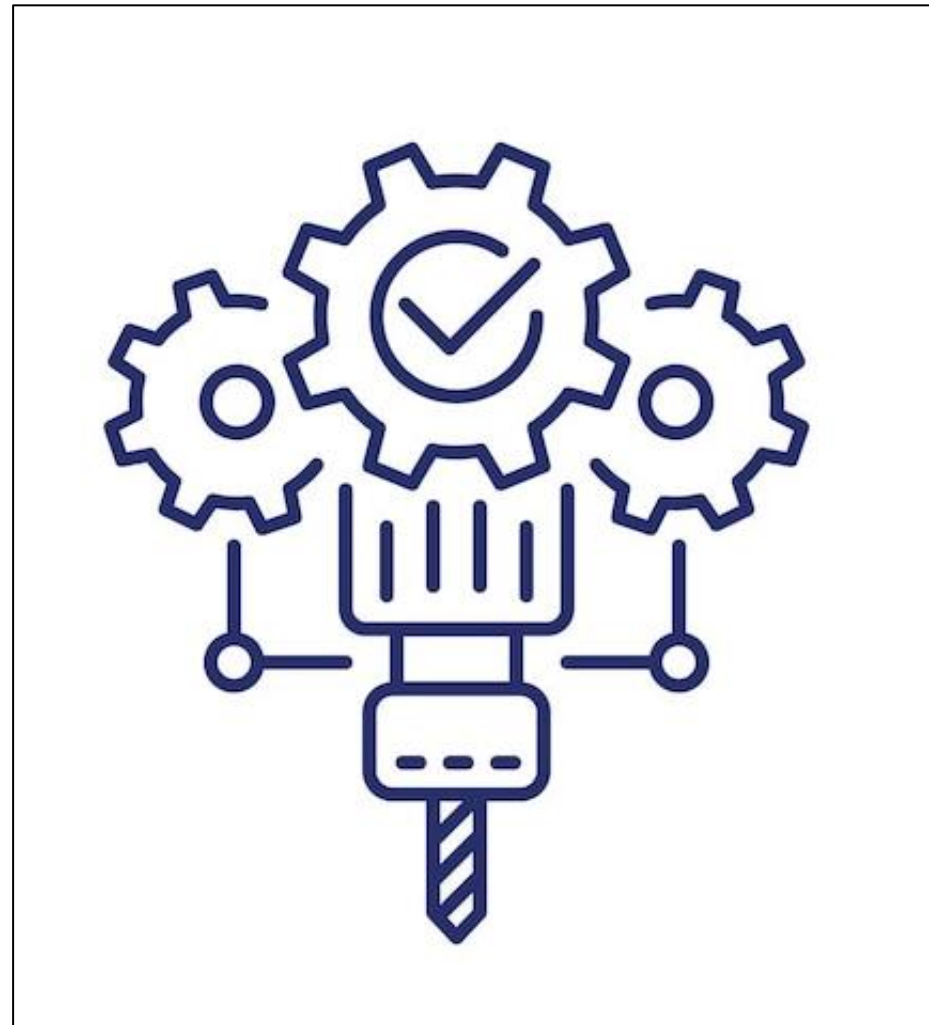
Express.js includes a built-in middleware function called `express.static()`.

```
Syntax:  
express.static(root, [options])
```

It uses `serve-static` to serve static files.

Parameters

The root parameter specifies the root directory from which static assets will be served.



return value: It yields an Object.

Example

Filename: index.js

```
var express = require('express');
var app = express();
var path = require('path');
var PORT = 3000;

// Static Middleware
app.use(express.static(path.join(__dirname, 'public')))

app.get('/', function (req, res, next) {
  res.render('home.ejs');
})

app.listen(PORT, function(err){
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

Example






Create a home.ejs file with the following code: home.ejs is the name of the file

```
<!DOCTYPE html>
<html>
<head>
  <title>express.static() Demo</title>
</head>
<body>
<h2>Greetings from GeeksforGeeks</h2>

</body>
</html>
```

Example

The project structure is given below:

 node_modules	06-06-2020 04:55 PM	File folder
 public	08-06-2020 11:25 AM	File folder
 views	07-06-2020 05:10 PM	File folder
 index.js	08-06-2020 11:23 AM	JS File
 package-lock.json	06-06-2020 04:55 PM	JSON File

Example

Ensure that the Express.js and ejs modules have been installed by running the following command:

```
npm install express  
npm install ejs
```

Example

Use the following command to run the index.js file:

```
node index.js
```


Output

```
Output:  
Server listening on PORT 3000
```

Output

Now, open the browser and navigate to `http://localhost:3000/` where the following output is obtained:



express.text()

`express.text()` is an Express.js built-in middleware function based on the `body-parser` and parses incoming requests payloads into strings.

```
Syntax:  
express.text([options])
```

This method returns the middleware that parses all JSON bodies as buffers and only considers requests where the content-type header matches the type option.

Properties

The following are the various properties of the options object available with this method:

Property	Purpose
inflate	It allows or prevents the handling of deflated or compressed bodies. True is the default value.
limit	It governs the request body's maximum size.
defaultCharset	If the charset is not specified in the request's content-type header, this option specifies the default character set for the text content.
type	It specifies the media type for the middleware to be parsed.

Example

Make a file called **expressText.js** and paste the following code into it. To run this code, use the command **node expressText.js** after creating the file.

```
// Importing the express module
var express = require('express');

// Initializing the express and port number
var app = express();
var PORT = 3000;

// Using the express.text middleware

// to convert into string
app.use(express.text());

// Reading content-type
app.post('/', function (req, res) {
  console.log(req.body)
  res.end();
})

// Listening to the port
app.listen(PORT, function(err){
  if (err) console.log(err);
  console.log("Server listening on PORT", PORT);
});
```

Example

Before calling the API endpoint:, set these properties:

1. In headers, set the content-type to text or plain.
2. In the POST request, include the following body - {name: Skillcart}

```
Output:  
C:\home  
ode>> node expressText.js  
Server listening on PORT 3000  
{  
  "title": "skillcart"  
}
```

`express.urlencoded()`

Express.js includes a built-in middleware function called `express.urlencoded()` and is based on the `body-parser` and parses incoming requests with `urlencoded` payloads.

```
Syntax:  
express.urlencoded([options])
```

This method returns middleware that parses all JSON bodies as buffers and only considers requests where the `content-type` header matches the `type` option.

Parameters

The options parameter contains properties such as extended, inflate, limit, verify, and so on.



Return value: Object

Example

Filename: index.js

```
var express = require('express');
var app = express();
var PORT = 3000;




app.use(express.urlencoded({extended:false}));

app.post('/', function (req, res) {
    console.log(req.body);
    res.end();
});

app.listen(PORT, function(err){
    if (err) console.log(err);
    console.log("Server listening on PORT", PORT);
});
```

Example

The project structure is given below:

Name	Date modified	Type	Size
 node_modules	06-06-2020 04:55 PM	File folder	
 index.js	06-06-2020 04:54 PM	JS File	0 KB
 package-lock.json	06-06-2020 04:55 PM	JSON File	14 KB

Output

POST a request to `http://localhost:3000/`. with the header 'content-type: application/x-www-form-urlencoded' set and body `{name: Skillcart}`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
PS C:\Users\Lenovo\Downloads\Geeksforgeeks Internship\NEW\Express> node index.js  
Server listening on PORT 3000  
[Object: null prototype] { '{ "name":"GeeksforGeeks" }': '' }  
█
```

The output given above is displayed on the console.

Working with Response Methods



Problem Statement:

Duration: 20 min.

You have been assigned a task to demonstrate the working of request methods.

Assisted Practice: Guidelines

Steps to be followed:

1. Install Postman in system for checks
2. Use `express.json()` response method in Express.js
3. Use `express.raw()` response method in Express.js
4. Use `express.Router()` response method in Express.js
5. Use `express.static()` response method in Express.js
6. Use `express.text()` response method in Express.js
7. Use `express.urlencoded()` response method in Express.js



Error Handling

Error Handling

This middleware is used to handle errors in Express.js



They are defined in the same way as other middleware functions, with an exception that error-handling functions must have four arguments rather than three - err, req, res, next.

Error Handling

For example, to respond to any error, use the following command:

```
app.use(function(err, req, res, next) {  
  console.error(err.stack);  
  res.status(500).send('Something broke!');  
});
```


Error Handling

People were previously dealing with errors in the routes themselves.

To separate the error logic



To send appropriate responses

Error Handling

The `next()` method from middleware takes you to the next middleware or route handler.



The `next(err)` function is used to handle errors.

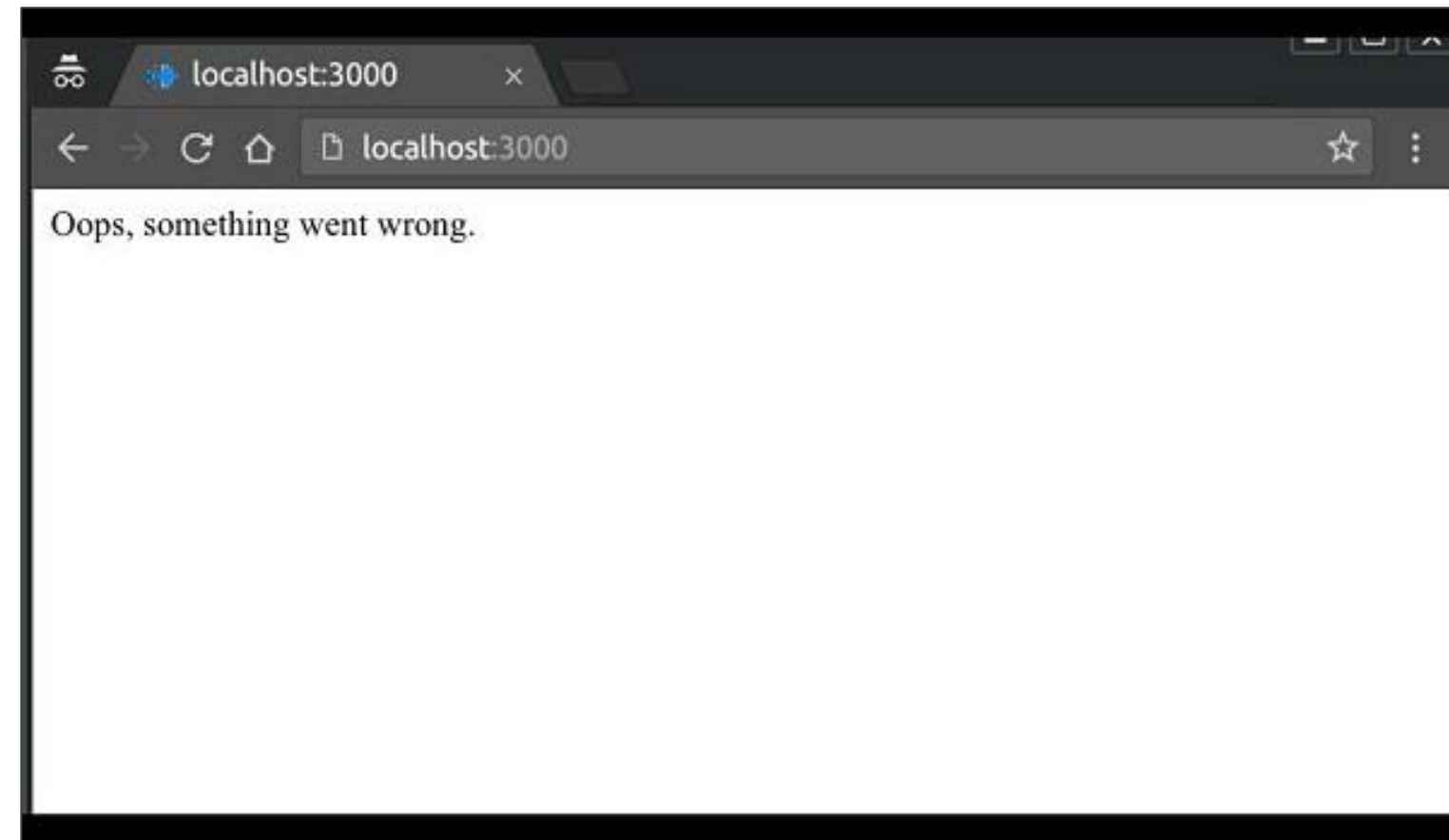
Error Handling

To illustrate Error-handling, consider the following example:

```
var express = require('express');var app =
express(); app.get('/', function(req, res){    //Create an
error and pass it to the next function    var err = new
Error("Something went wrong");    next(err);}); /* * other
route handlers and middleware here * .... */ //An error
handling middlewareapp.use(function(err, req, res, next) {
res.status(500);    res.send("Oops, something went
wrong.")}); app.listen(3000);
```

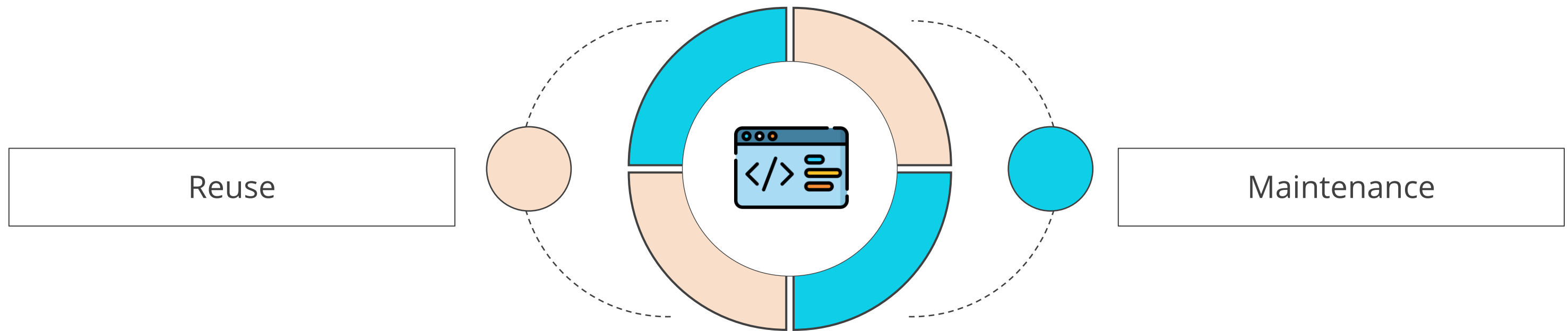
Error Handling

This error handling middleware can be strategically placed after routes or contain conditions.



Running an App

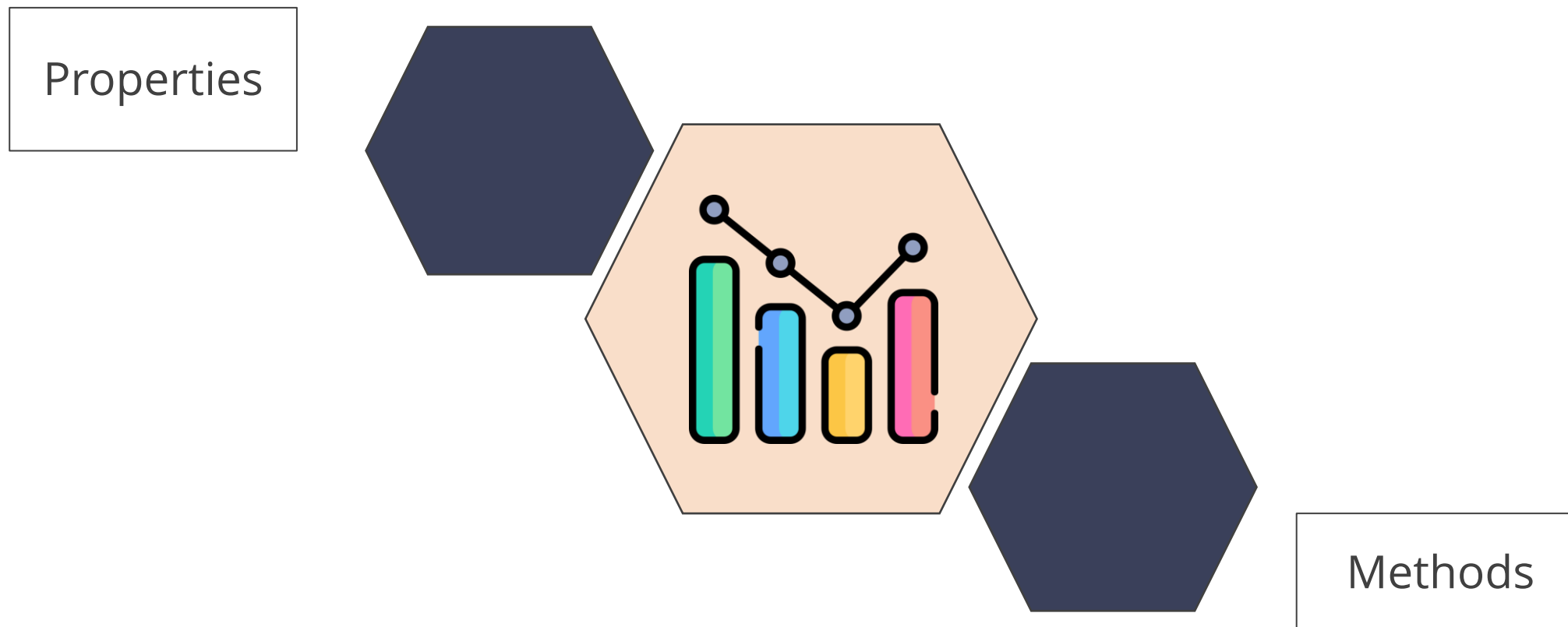
The Express.js class provides a few app-wide objects and methods on its object.



One can simply assign it once with `app.set('port', 3000);`. This allows them to set various application settings, including the port number on which the server will listen for incoming requests.

Running an App

To update the code later, modifications can be made in a single location.



app.locals

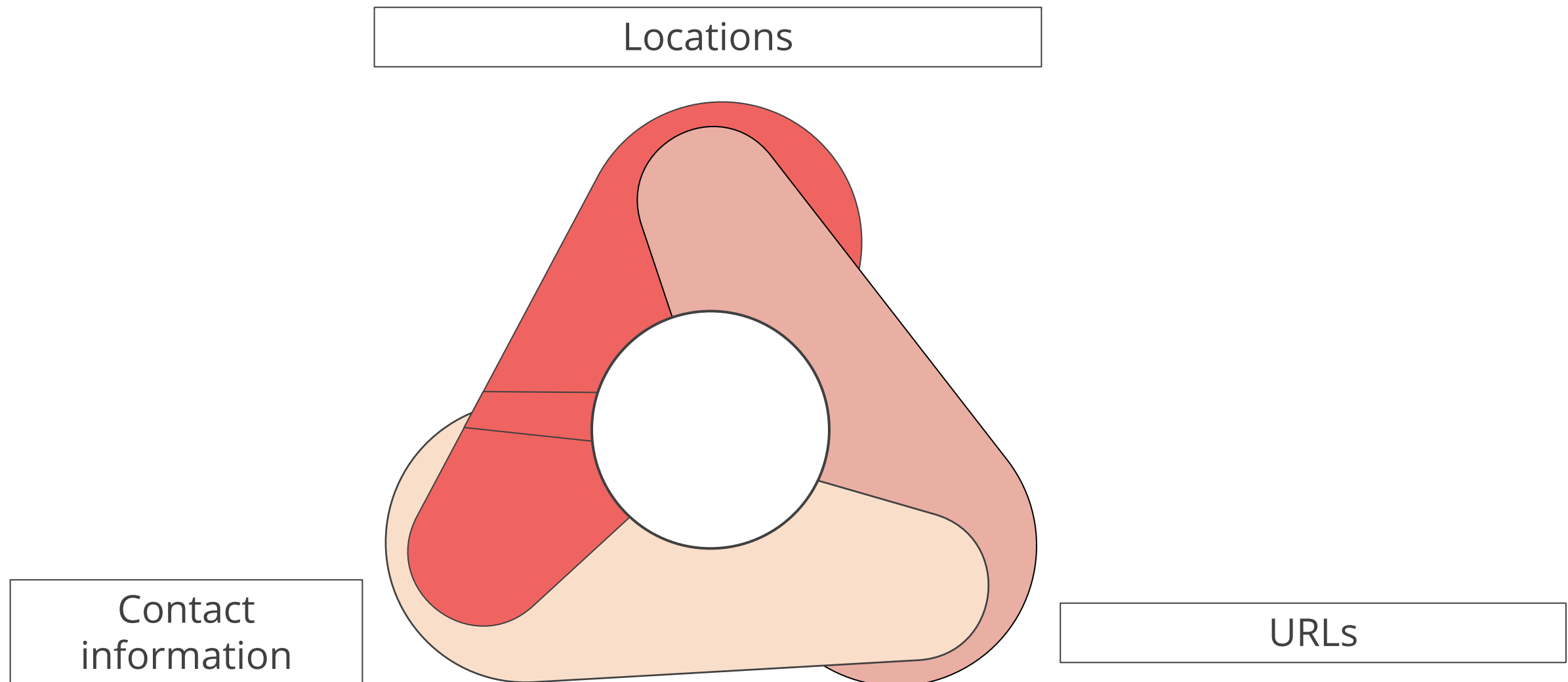
The app.locals object is similar to the res.locals object.



This object makes its properties available in all templates rendered by the app.

app.locals

Developers must exercise caution when using App.locals to reveal sensitive information.



app.locals

Example:

```
App.locals.lang = 'en';  
App.locals.appName = 'ABC';
```

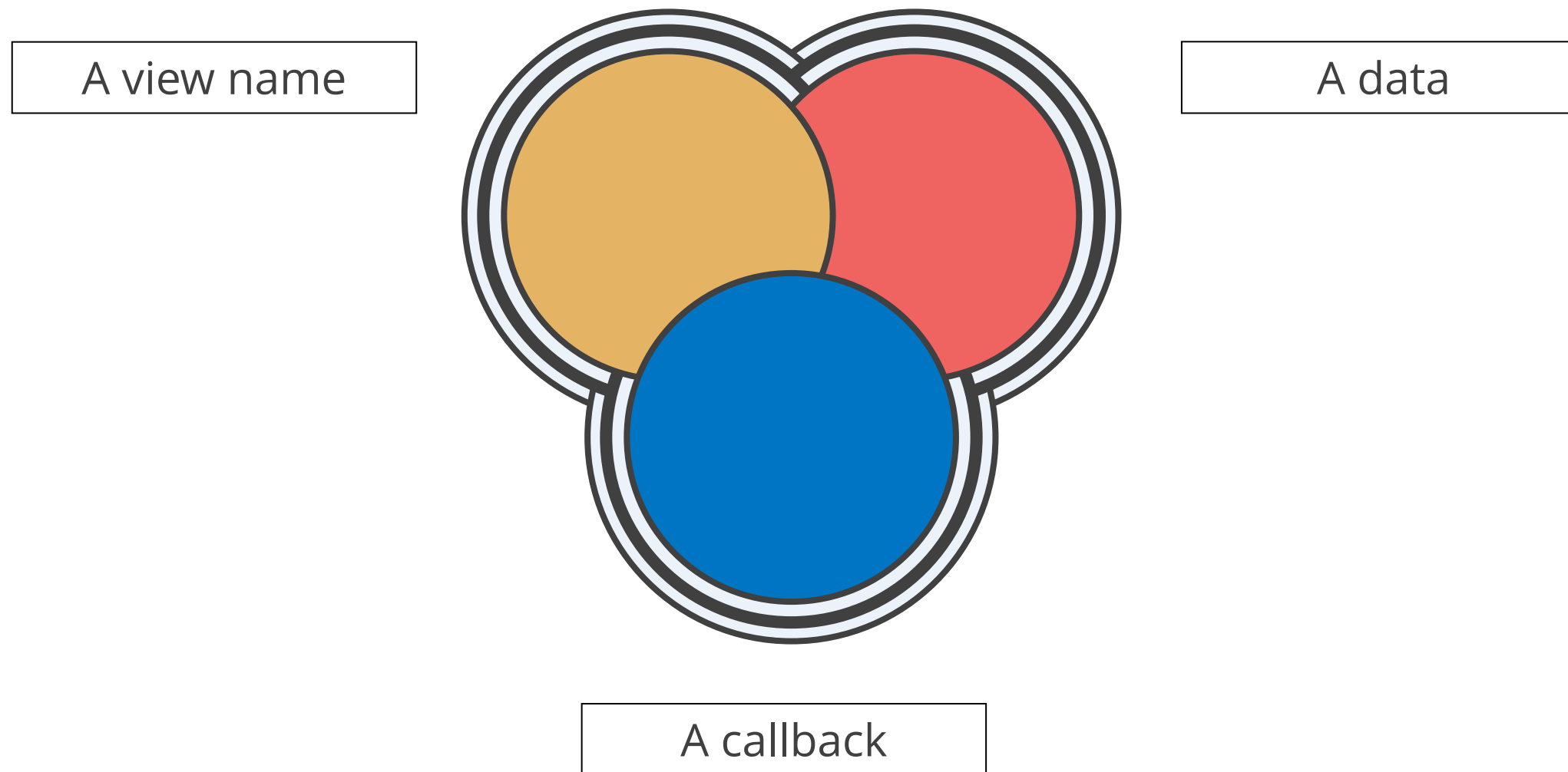
app.locals

The app.locals object can also be called as if it were a function:

```
App.locals([
  : 'XYZ',
  email:author 'XYZ@gmail.com',
  website: 'http://sampleexpressjs.com'
]);
```

app.render()

app.render() is called with either a view name or a callback.



app.render()

The system may have an email template for a **Thank you for signing up** message and another alternative for **Reset your password**:

```
var sendgrid = require('sendgrid')(api_user, api_key);
var sendThankYouEmail = function(userEmail) {
  app.render('emails/thank-you', function(err, html){
    if (err) return console.error(err);
    sendgrid.send({
      to: userEmail,
      from: app.get('appEmail'),
      subject: 'Thank you for signing up',
      html: html // The html value is returned by the app.render
    }, function(err, json) {
      if (err) { return console.error(err); }
      console.log(json);
    });
  });
};
```

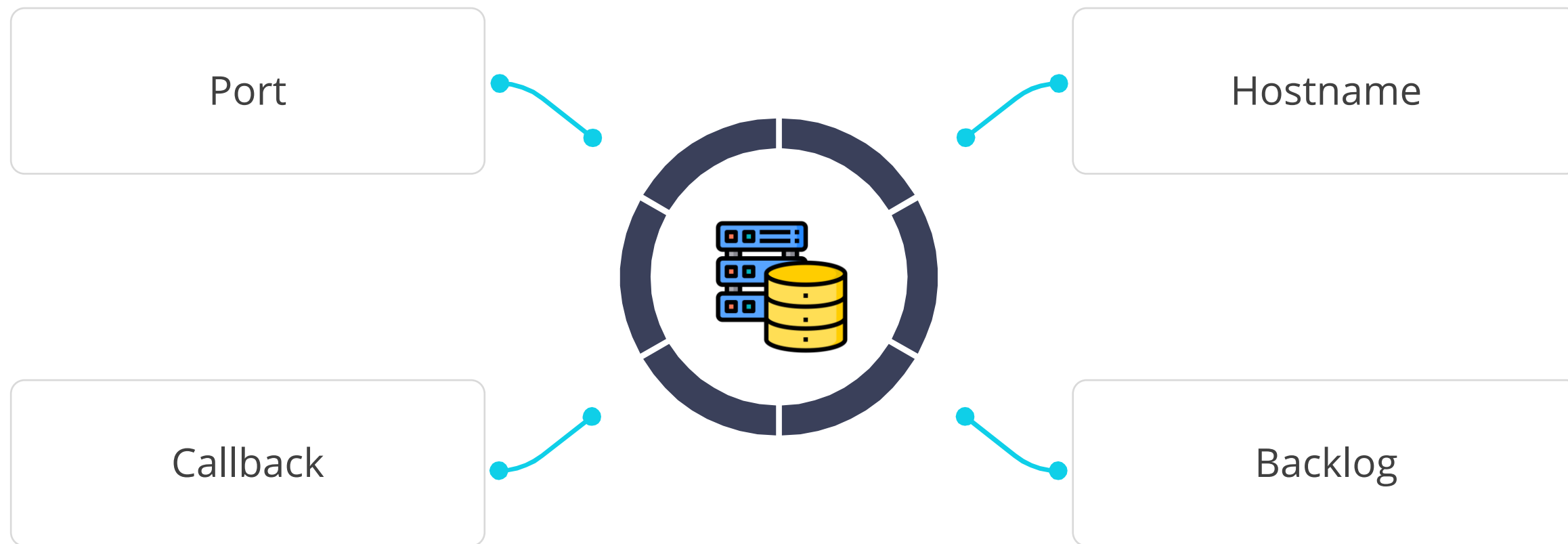
app.render()

The system may have an email template for a **Thank you for signing up** message and another alternative for **Reset your password**:

```
var resetPasswordEmail = function(userEmail) {  
  app.render('emails/reset-password', {token: generateResetToken()}, function(err, html) {  
    if (err) return console.error(err);  
    sendgrid.send({  
      to: userEmail,  
      from: app.get('appEmail'),  
      subject: 'Reset your password',  
      html: html  
    }, function(err, json) {  
      if (err) { return console.error(err); }  
      console.log(json);  
    });  
  });  
};
```

app.listen()

The listen() method is similar to the server.listen() method from the Node.js http module.
Following are some common parameters between the both:



app.listen()

This is one of the methods for launching an Express.js application.



This method gets the server ready to receive incoming HTTP requests on a specified port, launching the Express application to handle and respond to the user requests.

app.listen()

The backlog is the total number of pending connections that are queued. The queued connections are the incoming requests when coming from multiple clients.



app.listen()

Example: The launch of Express.js app on a specific port (3000):

```
var express = require('express');  
var app = express();  
// ... Configuration  
// ... Routes  
app.listen(3000);
```

app.listen()

The Express.js generator generated this approach in the ch2/hello.js and ch2/hell-name.js files.

```
module.exports = app
```

app.listen()

Also, don't run the app.js file with `$ node app.js`.

```
#!/usr/bin/env node
```

Use `$/bin/www` to run the shell script www.

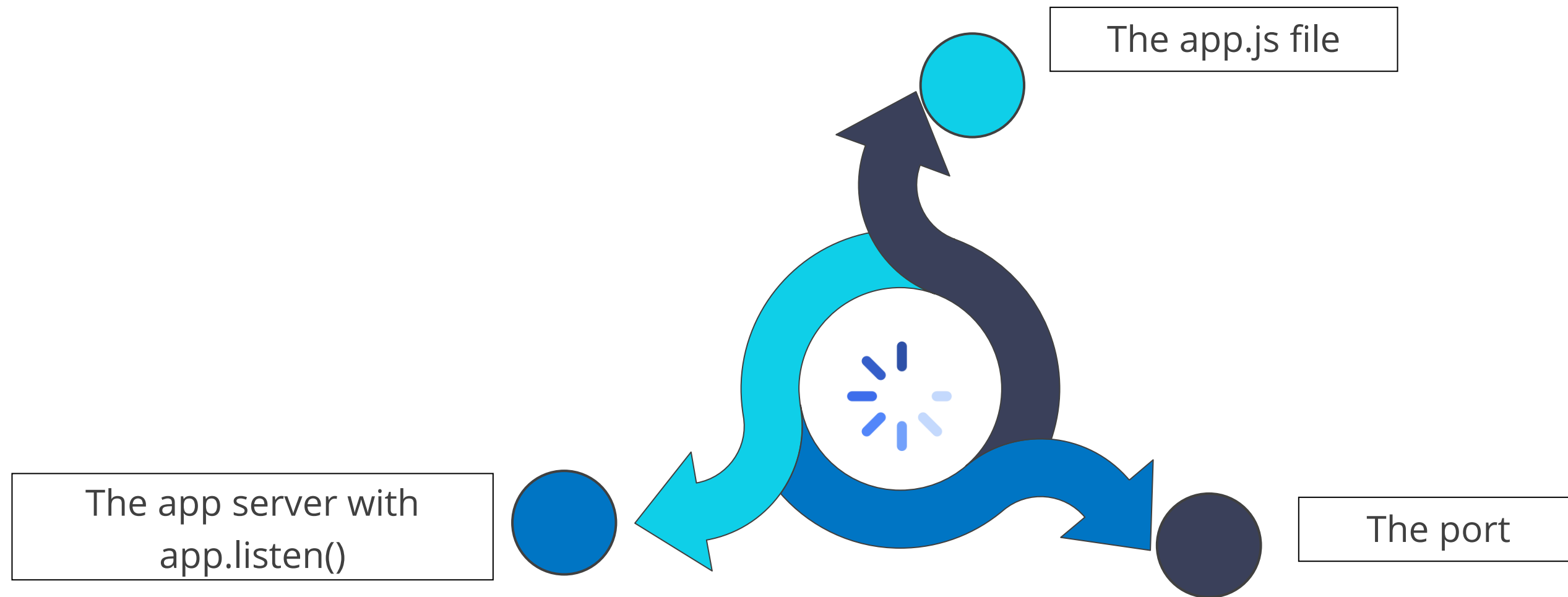
app.listen()

The preceding line converts the shell script into a Node.js program.

```
var debug = require('debug')('cli-app');
var app = require('../app');
app.set('port', process.env.PORT || 3000);
var server = app.listen(app.get('port'), function() {
  debug('Express server listening on port ' +
  server.address().port);
});
```

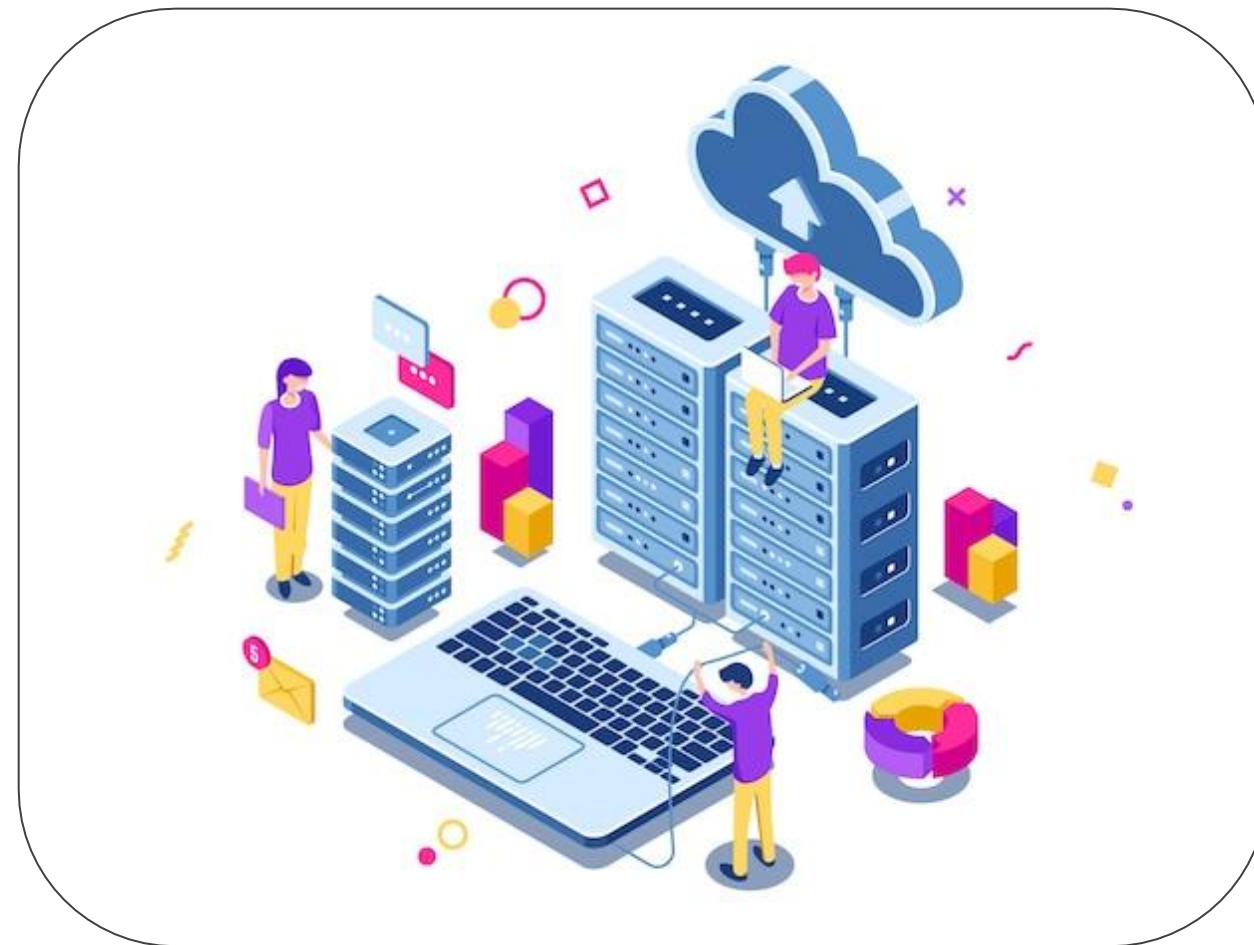
app.listen()

This program loads the app object from:



app.listen()

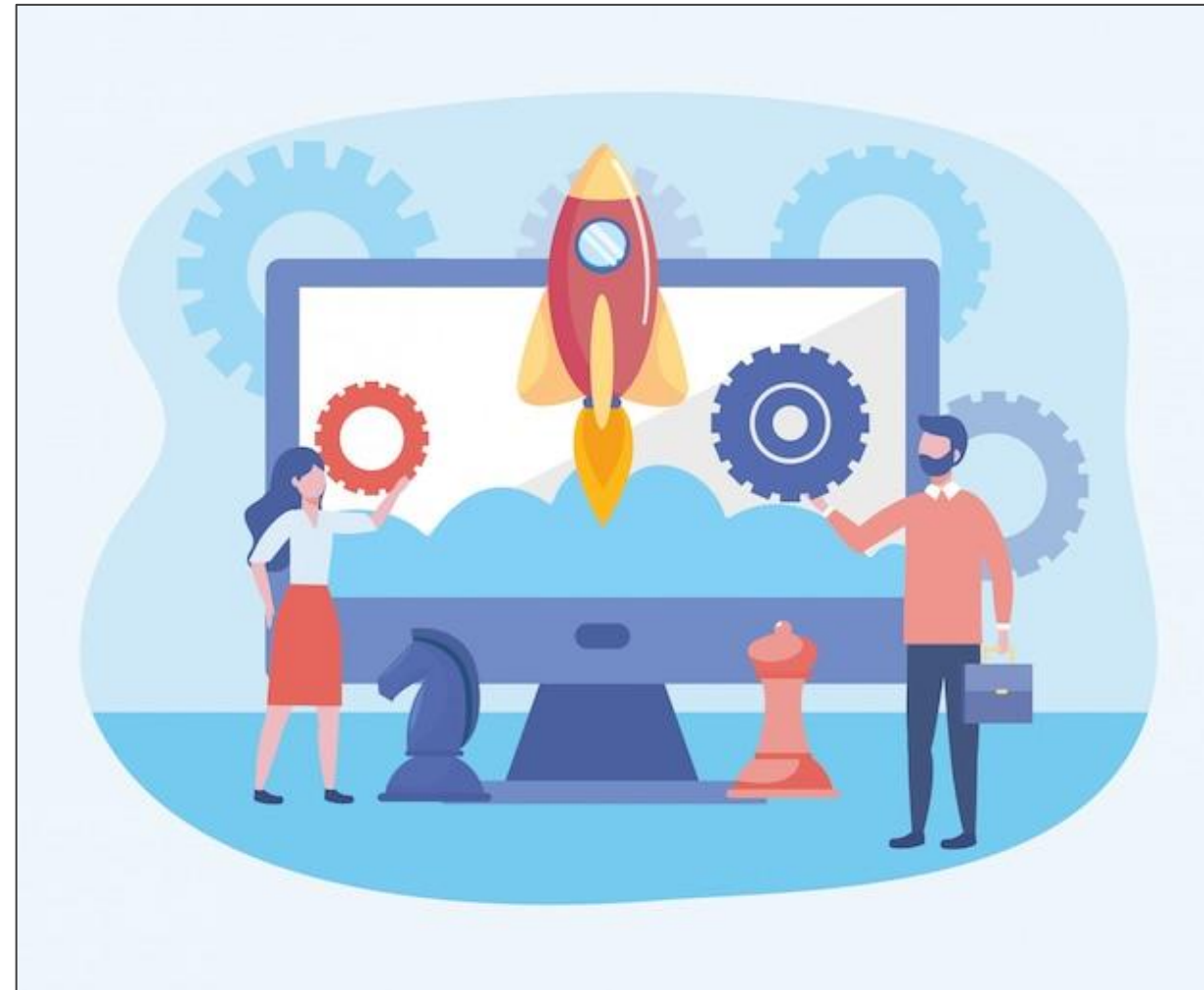
The main server file in the preceding example (ch2/cli-app/app.js)



There is no way to start the server with `$ node app` because the object has been exported.

app.listen()

When you want to launch the server but still want to export it, when necessary, use `app.listen()`.



app.listen()

Use the approach to determine whether a module is a dependency with `require.main === module` condition. If that's the case, one will launch the application.

```
var server = http.createServer(app);
var boot = function () {
  server.listen(app.get('port'), function(){
    console.info('Express server listening on port ' + app.get('port'));
  });
  var shutdown = function() {
    server.close();
  }
  if (require.main === module) {
    boot();
  } else {
    console.info('Running app as a module');
    exports.boot = boot;
    exports.shutdown = shutdown;
    exports.port = app.get('port');
  }
}
```

If it isn't, one will expose the methods and the app object.

app.listen()

The application of the Express.js app to the core Node.js server function can help start a server.



app.listen()

This is useful for launching both an HTTP server and an HTTPS server from the same code base:

```
var express = require('express');
var https = require('https');
var http = require('http');
var app = express();
var ops = require('conf/ops');
//... Configuration
//... Routes
http.createServer(app).listen(80);
https.createServer(ops, app).listen(443);
```

Error Handling Commands



Problem Statement:

You have been assigned a task to demonstrate the use of error handling commands.

Duration: 10 min.

Assisted Practice: Guidelines

Steps to be followed:

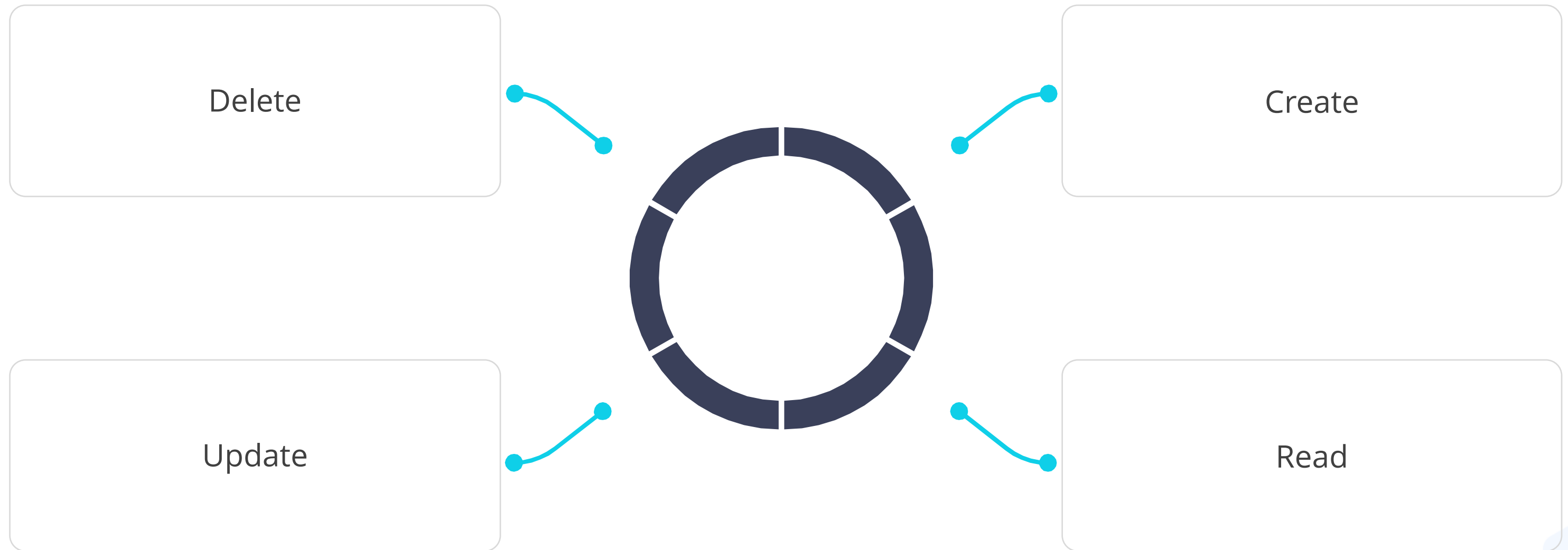
1. Install the postman in system for checks
2. Use `app.routes` in `Express.js`
3. Use `app.locals` in `Express.js`
4. Use `app.render()` in `Express.js`
5. Use `app.listen()` in `Express.js`



app.set() and app.get()

CRUD Operation

It is defined as a group of operations that servers execute; the four basic operations being:



CREATE (POST)

It is defined as an operation that is used to create a new resource of a specific type. HTTP POST method is used to perform the create operation.

```
router.post('/', async(req,res) => {  
  const course = new Course({  
    name: req.body.name,  
    tech: req.body.tech,  
  })
```

READ (GET)

It is defined as an operation that is used to retrieve or read information from the server. HTTP GET method is used to perform the read operation.

```
router.get('/:id', async(req,res) => {  
  try{  
    const course = await Course.findById(req.params.id)  
    res.json(course)  
  }catch(err){  
    res.send('Error ' + err)  
  }  
})
```


UPDATE

It is defined as an operation that is used to update information from the database.
The HTTP PUT or PATCH methods are used to perform the update operation.

```
router.patch('/:id', async (req, res) => {  
  try {  
    const course = await Course.findById(req.params.id)  
    course.name = req.body.name  
    const a1 = await course.save()  
    res.json(a1)  
  } catch (err) {  
    res.send('Error')  
  }  
  
})
```

DELETE

It is defined as an operation that is used to delete a resource from the system. HTTP DELETE method is used to perform the delete operation.

```
router.delete('/:id', async (req, res) => {  
  try{  
    const course = await Course.findById(req.params.id)  
    const a1 = await course.remove()  
    res.json("Deleted")  
  } catch (err) {  
    res.send('Error')  
  }  
  
})
```

CRUD Operations



Problem Statement:

You have been assigned a task to perform CRUD operations.

Duration: 20 min.

ASSISTED PRACTICE

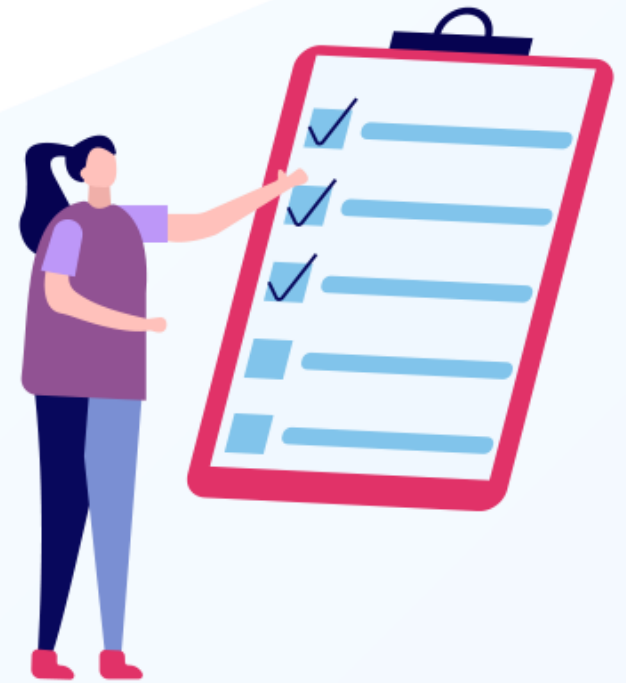
Assisted Practice: Guidelines

Steps to be followed:

1. Install prerequisites for performing CRUD operations
2. Analyze the output of CRUD operations

Key Takeaways

- 🕒 The Response object (res) specifies the HTTP response that an Express.js app sends when it receives an HTTP request.
- 🕒 `express.json()` is an Express.js built-in middleware function, and this method is used to parse incoming requests with JSON payloads.
- 🕒 The Express.js class provides a few app-wide objects and methods on its object.
- 🕒 Create is defined as an operation that is used to create a new resource of a specific type.





Thank You