

Express

INTRO

What is Express ?

URL Routing

Middleware

Template Engine

What is Express ?

A web application framework for Node

- Minimal and flexible
- Great for building Web APIs
- Popular services built on Express i.e. **MySpace**, **Ghost** and more
- Foundation for other tools and frameworks, like Kraken and Sails

Why Express ?

A web application framework for Node

Express is a light-weight web application framework to help organize your web application into an **MVC** architecture on the server side . You can use a variety of choices for your templating language (like EJS, Jade, and Dust.js).

You can then use a database like **MongoDB** with **Mongoose** (for modeling) to provide a backend for your Node.js application . Express.js basically helps you manage everything, from **routes**, to **handling requests** and **views**.

Installing Express ?

Use **npm** to install the latest stable version

```
$ npm install express
```

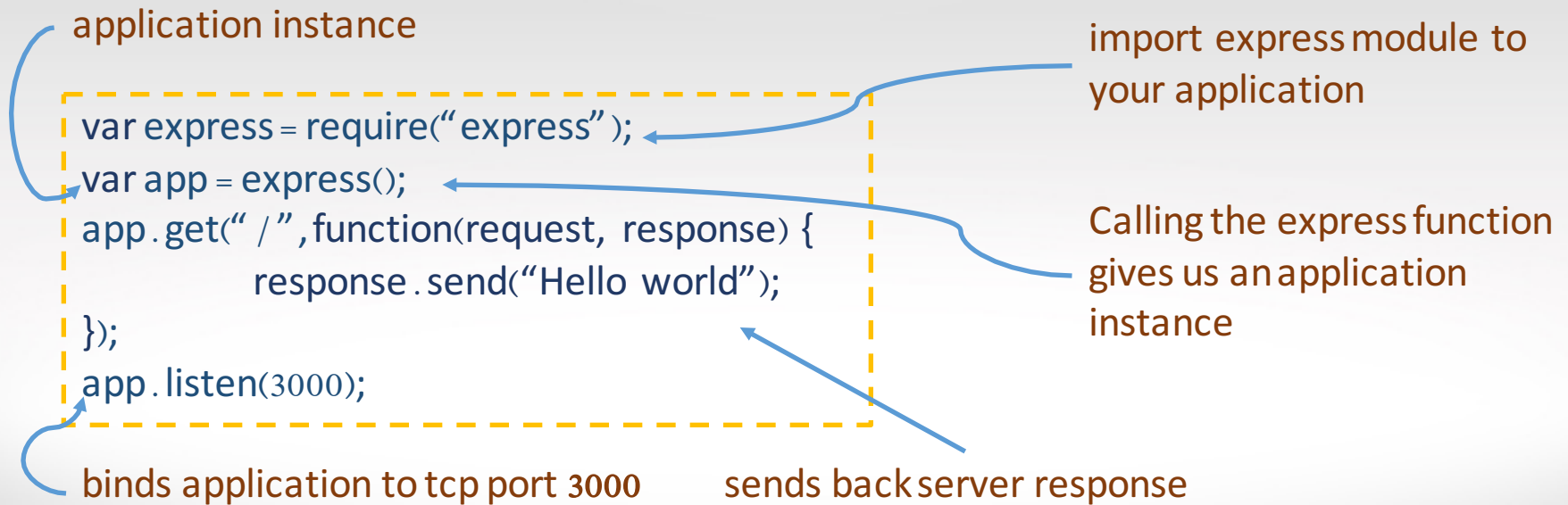
```
$ npm install express@4.9
```

installs latest version from the 4.9 branch

```
$ npm install express@3.15.2
```

installs specific version

Hello World



The Request and Response objects



The Request and Response objects

```
app.get("/", function(request, response) {  
    response.write("Hello world");  
    response.end();  
});
```

← `response.send("Hello world");`

← `response.json("Hello world");`

To redirect request

`response.redirect("url")`

Serving files with `sendFile`

`response.sendFile`
`(__dirname+"/public/index.html")`

`response.status(404)`



URL Routing

```
var express = require("express");
var app = express();
app.get("/", function(request, response) {
    response.send("Hello world");
});
app.get("/users", function(request, response) {
});
app.get("/contact", function(request, response) {
});
app.post("/contact", function(request, response) {
    response.send("Hello world");
});

app.listen(3000);
```

http://localhost:3000/users



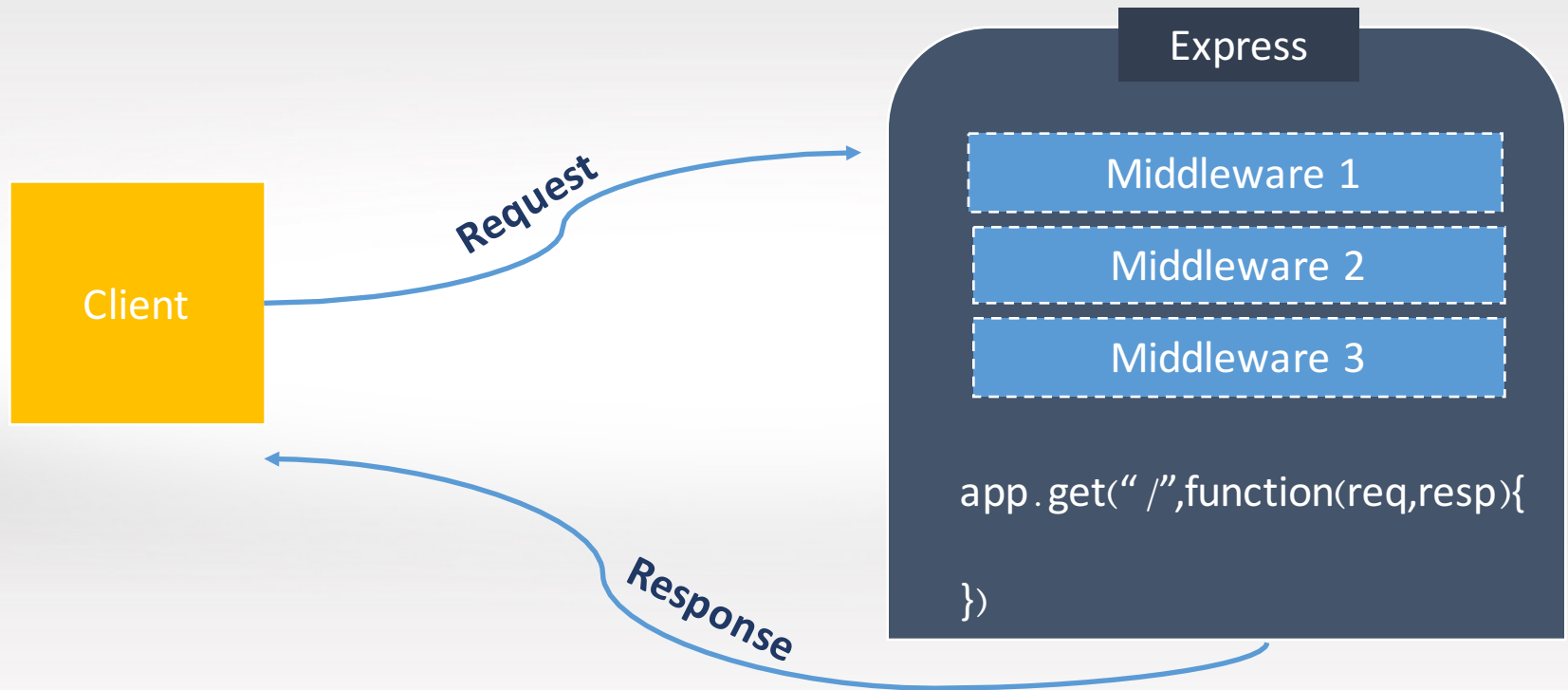
http://localhost:3000/contact



Middleware

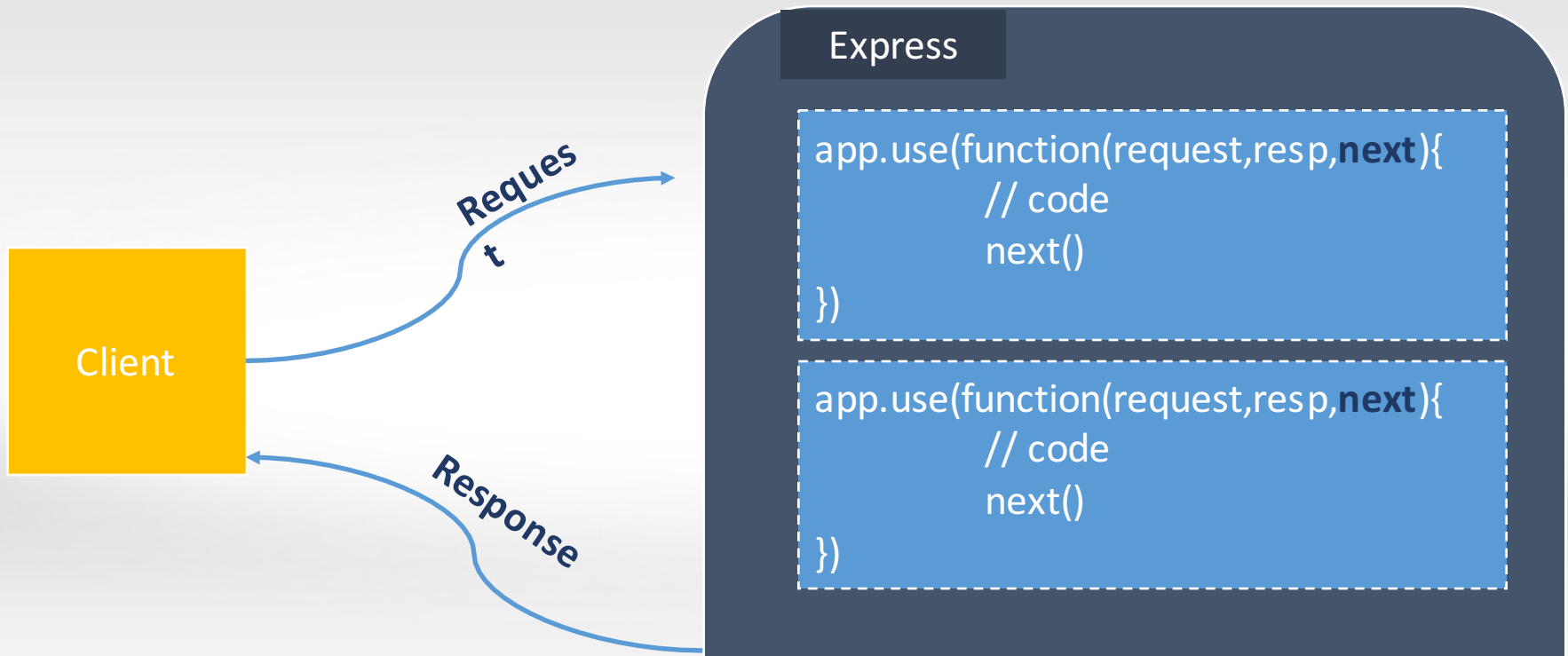
Functions executed **sequentially** that access **request** and **response**

`app.use()` function , to add middleware



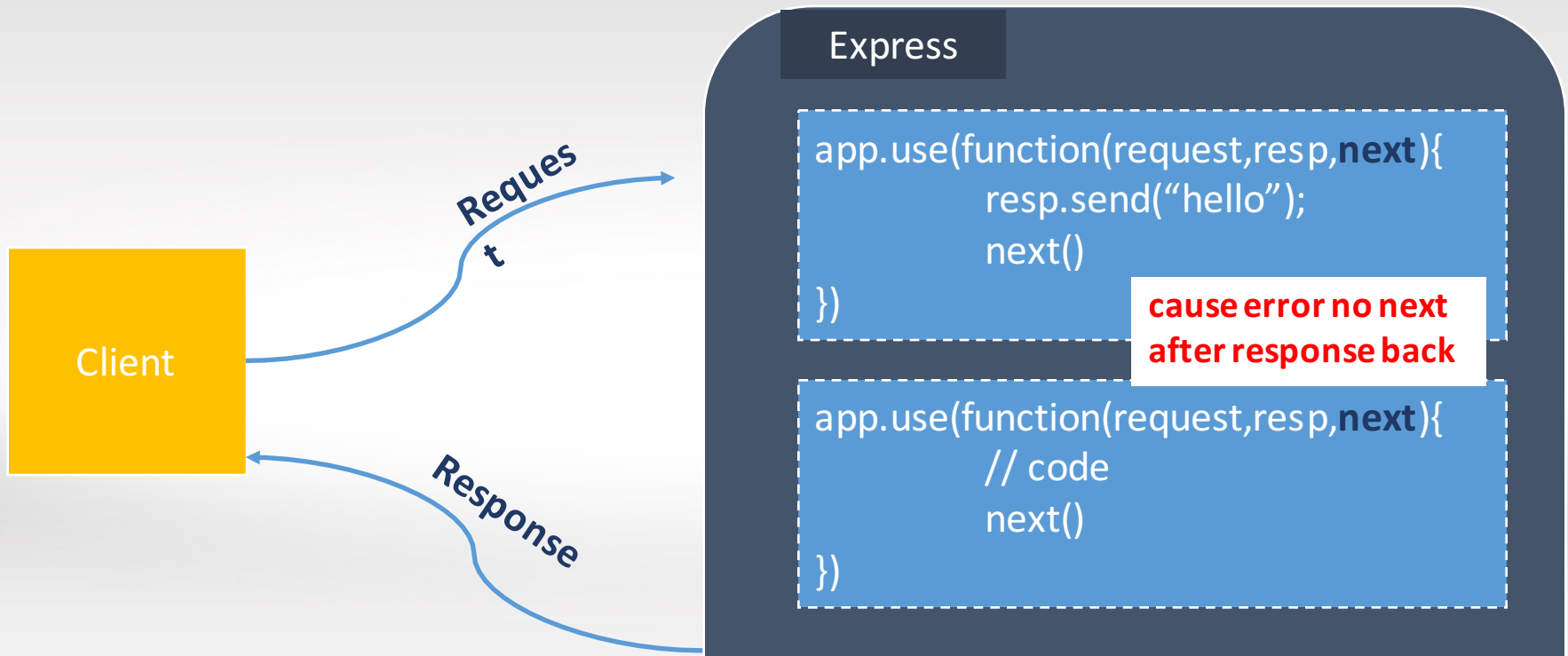
Middleware

When **next** is called, processing moves to the next middleware.



Middleware

The flow **stops** once the response is sent back to the client



Static Middleware

The static middleware serves **everything** under the specified folder



Express

```
app.use(express.static('public'));
```

```
<!DOCTYPE html>
...
<body>
  <h1>Blocks</h1>
  <p><img src='blocks.png'></p>
</body>
</html>
```

User Params

Reading from the URL, Reading query string parameters

```
var express = require("express");  
var app = express();  
app.get("/users", function(request, response){  
    response.send(request.query.name);  
});  
app.listen(3000);
```

http: //localhost:3000 /users /?name="ahmed"

A blue arrow originates from the query string portion of the URL, specifically from the value "ahmed" enclosed in quotes, and points towards the `request.query.name` property access in the code block above.

Creating Dynamic Routes

Placeholders can be used to name arguments part of the **URL path**

```
var express = require("express");
var app = express();
app.get("/users/:name", function(request, response){
    response.send(request.params.name);
});
app.listen(3000);
```



http: //localhost:3000 /users /ahmed

app.param

The **app.param** function maps placeholders to callback functions. It's useful for running **pre-conditions** on dynamic routes.

```
var express = require("express");
var app = express();
app.param("name", function(request, response, next) {
    request.newName = request.params.name + "new";
});

app.get("/users/:name", function(request, response) {
    response.send(request.newName);
});

app.get("/users/:name", function(request, response) {
    response.send(request.newName);
});

app.listen(3000);
```


Post Requests

Post requests send data in request body so that we need to install module and add it as middleware to parse request body

```
npm install body-parser
```

```
<form action="/contact" method="post">  
  <input type="text" name="name" />  
  <input type="text" name="message" />  
  <input type="submit" value="send" />  
</form>
```

Post Requests

Post requests send data in request body so that we need to install module and add it as middleware to parse request body

```
var express = require("express");
var app = express();
var bodyParser = require("body-parser");
var parseUrlencoded = bodyParser.urlencoded({extended: false});

app.post("/contact",parseUrlencoded,function(request,response){
    var name=request.body.name;
    var message=request.body.message;
});
app.listen(3000);
```

Route Instances

Repetition in route names

All routes seem to be handling requests to similar **paths** . . .

```
var express = require("express");
var app = express();
app.get("/contact", function(request, response) {
});
app.post("/contact", function(request, response) {
});
app.get("/contact/:name", function(request, response) {
});
app.delete("/contact/:name", function(request, response) {
});
app.listen(3000);
```

identical path



identical path

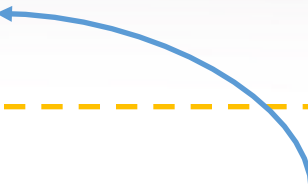


Route Instances

Replacing repetition with a route instance

Using **app.route** is a recommended approach for avoiding **duplicate** route names

```
var express = require("express");  
var app = express();  
var contactRoute = app.route("/contact")  
app.listen(3000);
```



returns route object which handles
all requests to the **/contact** path

Route Instances

Replacing repetition with a route instance

Using **app.route** is a recommended approach for avoiding **duplicate** route names

```
var express = require("express");
var app = express();
var contactRoute = app.route("/contact")
contactRoute.get(function(req,resp){
})
contactRoute.post(function(req,resp){
})
app.listen(3000);
```

Chaining function calls on route

Chaining functions can eliminate **intermediate variables** and help our code stay more **readable**. This is a pattern commonly found in Express applications.

```
var express = require("express");  
var app = express();  
app.route("/contact")  
.get(function(req,resp){  
})  
.post(function(req,resp){  
})  
app.listen(3000);
```

chaining means calling functions on the return value of previous functions


lines starting with **dot** indicate function calls on the object returned from the **previous line**

Route File

Single application file is too long , Too many lines of code in a text file is a **bad smell**

our **app.js** file is growing too long. **Extracting routes to modules** This helps clean up our code and allows our main **app.js** file to easily accommodate additional routes in the future .

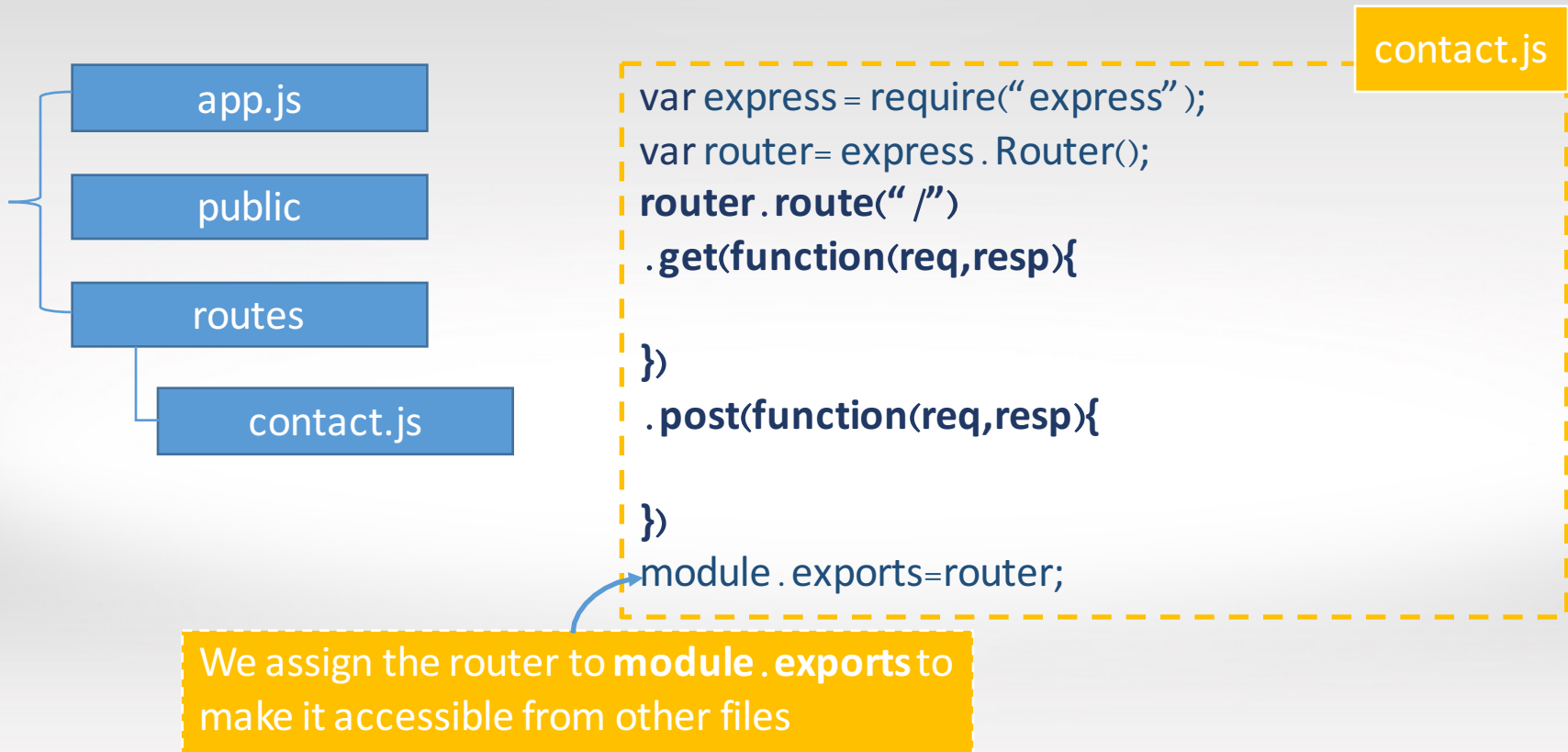
```
var express = require("express");  
var app = express();  
app.route("/contact")  
var contact=require(". /routes /contact");  
app.use("/contact",contact);  
app.listen(3000);
```



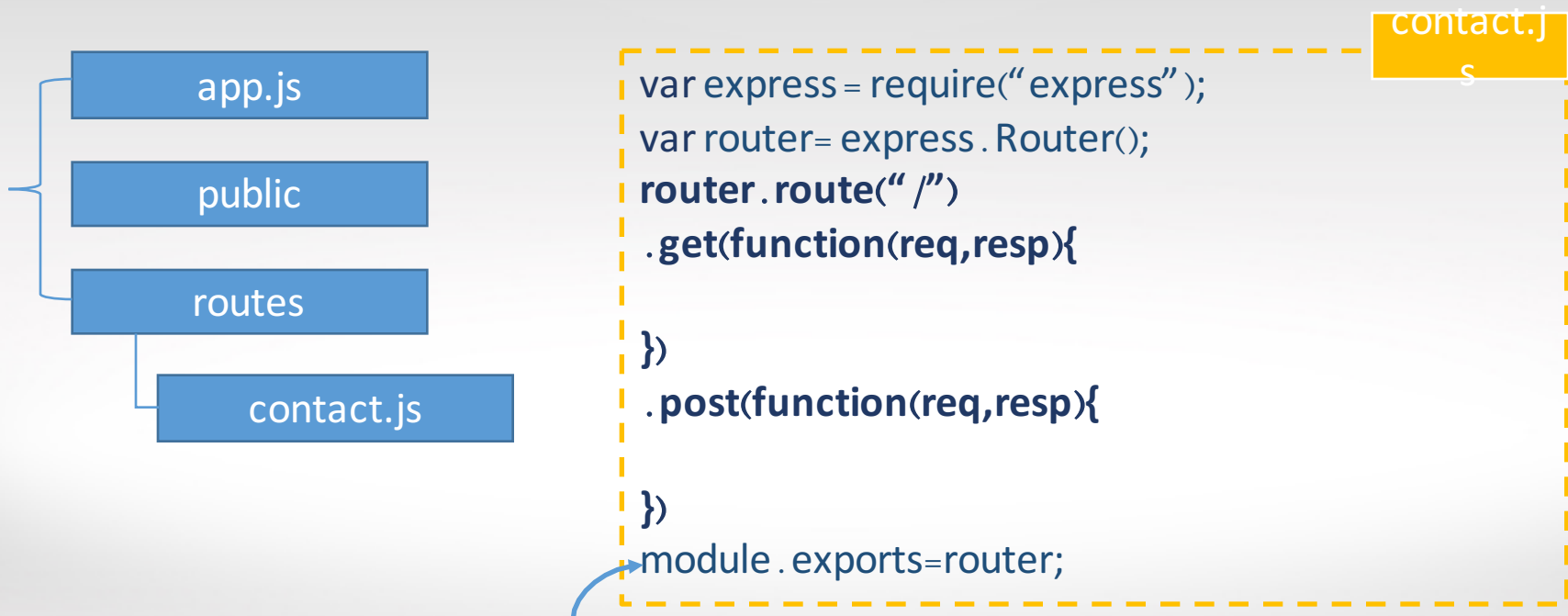
we'll move our routes to this new file

router is mounted in a particular **root url**

Route File



Route File



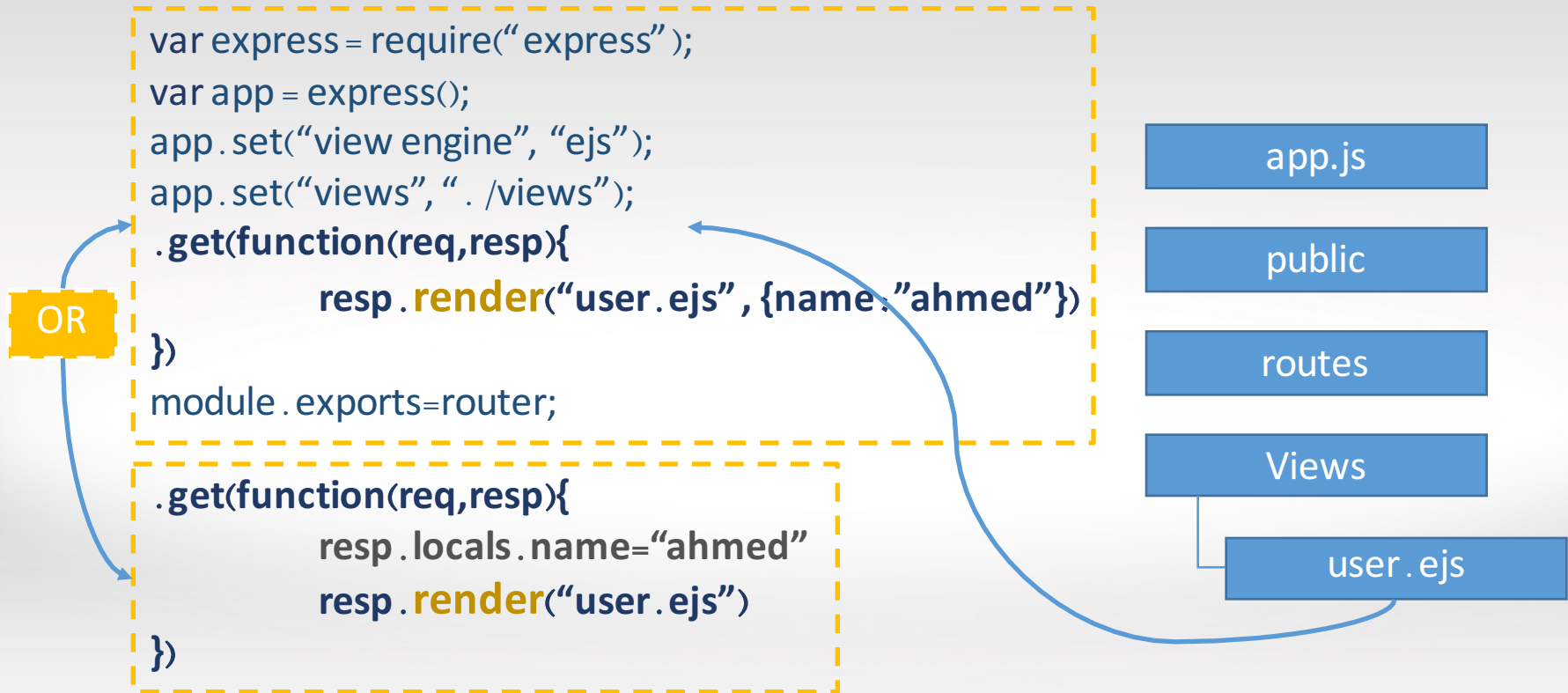
We assign the router to `module.exports` to make it accessible from other files

Template Engines (Views)

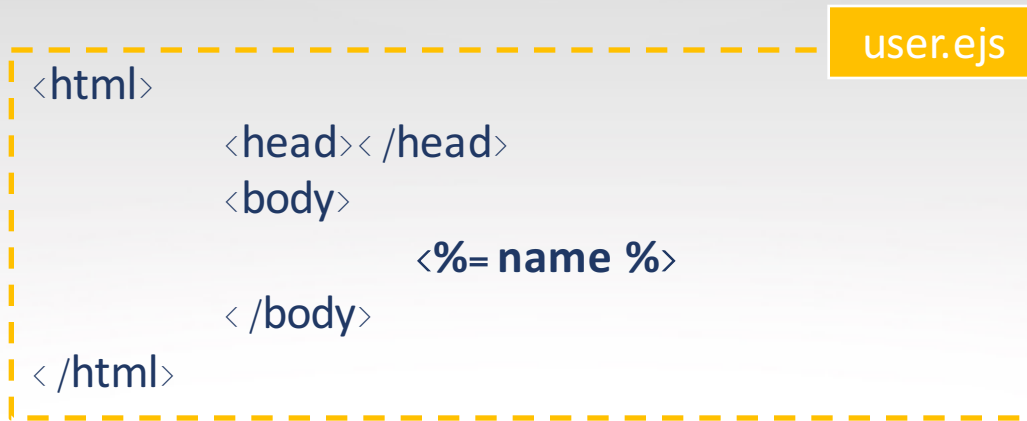
A template engine enables you to use static template files in your application . At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client . This approach makes it easier to design an HTML page . Some popular template engines that work with Express are **Pug**, **Mustache**, and **EJS** . The Express application generator uses Jade as its default, but it also supports several others .

```
npm install ejs
```

Template Engines (Views)



Template Engines (Views)



The diagram illustrates a template engine view. It features a dashed orange rectangular box containing HTML-like code. To the top right of this box is a yellow rectangular label with the text 'user.ejs'. The code inside the box is as follows:

```
<html>  
  <head></head>  
  <body>  
    <%= name %>  
  </body>  
</html>
```

Template Engines (Views)

Too pass array from app.js to template and render it

```
resp.locals.users=[  
  {name:"ahmed"},  
  {name:"ali"}  
]  
resp.render("user.ejs")
```

app.js

```
<html>  
  <head></head>  
  <body>  
    <% users.forEach(function(user){ %>  
      <h1><%= user.name %></h1>  
    <% }) %>  
  </body>  
</html>
```

user.ejs

USING EJS PARTIALS

Like a lot of the applications we build, there will be a lot of code that is reused. We'll call those

index.ejs

```
<html>
  <head></head>
  <body>
    <%include ../partials/header%>
    <!-- content code here-->
    <%include ../partials/sidebar %>
    <%include ../partials/footer %>
  </body>
</html>
```

header.ejs

footer.ejs

sidebar.ejs

Create Logger Middleware



Code Time



Thank You

E-mail Address :

ahmedcs2012@gmail.com