Templates -View Engine(pug)

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**

Pug is a very powerful templating engine which has a variety of features including **filters, includes, inheritance, interpolation.**

To use Pug with Express, we need to install it,

```
npm install pug
```

Now that Pug is installed, set it as the templating engine for your app. You **don't** need to 'require' it. Add the following code to your **.js** file.

```
app.set('view engine', 'pug');
app.set('views','_dirname);
```

Now create a file called **first_view.pug**, and enter the following data in it.

```
doctype html
html
  head
    title = "Hello Pug"
  body
    p people Hello World!
```

To run this page, add the following route to your app −

```
app.get('/', function(req, res){
  res.render('first_view'); // this will display your pug file on browser
});
```

Attributes

To define attributes, we use a comma separated list of attributes, in parenthesis. Class and ID attributes have special representations. The following line of code covers defining attributes, classes and id for a given html tag.

```
div.container.column.main#division(width = "100", height = "100")
```

This line of code, gets converted to the following. −

```
<div class = "container column main" id = "division" width = "100" height =
"100"></div>
```

Example:

Simple example to understand the concept of pug

```
html

head

title PUG example

body

h1 LJU

p lets learn pug

i Thank you


// apply style or attribute

h5(style="text-transform:uppercase;color:pink",align=”center”) hello world


//to write multiple line in same tag. but it will print in one line

p hello world

|The file will not

|render properly if the

|programmer does not make

|sure of proper indentation

// create list

ul

li Mango

li Watermelon

li Pineapple
```

```
// to create table

table(border='1px red solid')

        tr

        th name

        th id

        tr

        td abc

        td 1
```

Passing Values to Templates

When we render a Pug template, we can actually pass it a value from our route handler, which we can then use in our template. Create a new route handler with the following.

```
var express = require('express');
var app = express();
app.set("view engine","pug")
app.set("views",__dirname)
app.get('/', function(req, res){
  res.render('dynamic', { name: "learning", url:"http://www.gmail.com" });
});

app.listen(3000);
```

And create a new file in same directory, called **dynamic.pug**, with the following code −

```
html
  head
    title=name
  body
    h1 #{name}  //fetch value of name  from js file
    a(href = url) URL  // fetch value of url from js file
```

Include

Pug provides a very intuitive way to create components for a web page. For example, if you see a news website, the header with logo and categories is always fixed. Instead of copying that to every view we create, we can use the **include** feature. Following example shows how we can use this feature Create 3 views with the following code −

HEADER.PUG

```
div.header.
  I'm the header for this website.
```

CONTENT.PUG

```
html
  head
    title Simple template
  body
    include ./header.pug
    h3 I'm the main content
    include ./footer.pug
```

FOOTER.PUG

```
div.footer.
  I'm the footer for this website.
```

Create a route for this as follows –

App.js

```
var express = require('express');
var app = express();
app.set("view engine","pug")
app.set("views",__dirname)
app.get('/', function(req, res){
   res.render('content'); // here content is pug file name
});

app.listen(3000);
```

External Stylesheet

To apply external stylesheet , first you have to create style.css file ,style.pug and style.js file Then use include command to link external stylesheet in your pug file.

Style.css

```
    #c1

    { color:red;

    }
```

Style.pug

```
Html
    Head
        Style
            Include ./style.css
        Body
            H1#c1 Hello world
```

Style.js

```
var express = require('express');
var app = express();
app.set("view engine","pug")
app.set("views",__dirname)
app.get('/', function(req, res){
   res.render('style');     // it will render style.pug file
});

app.listen(3000);
```

Example

Main.pug

```
html
head
title hello world
body
h1(align="center") pug example
p(style="color:red;font-size:50px") lets leran example
i thanks
```

main.js

```
const expr=require("express");
const app=expr();
app.set("view engine","pug");
app.get("/",(req,res)=>
{
    res.render(__dirname+"/main");
});

app.listen(3000,()=>
```

```
{
    console.log("server start");
});
```

Program 1: Create one pug file which contains a text field and a password field. By submitting the form, on next page called “/login” submitted data will be displayed

Form.pug

```
html
  head
    title Login Form
  body
    h1 Login
    form(action="/login" method="post")
      label Username:
      input(type="text" id="username" name="username" required)
      br
      label Password:
      input(type="password" id="password" name="password" required)
      br
      input(type="submit" value="Login")
```

display.pug

```
html
  head
    title Login Data
  body
    h1 Welcome, #{username}!
    p Your username is: #{username}
    p Your password is: #{password}
```

pug.js

```
const express = require('express');
const app = express();
var bp=require("body-parser");
var url=bp.urlencoded();
```

```
app.set('view engine', 'pug');
//app.set('views', __dirname);
app.get('/', (req, res) => {
  res.render(__dirname +'/form');});
app.post('/login',url, (req, res) => {
  const { username, password } = req.body;
  res.render(__dirname +'/display', { username: username, password:
password });
});
app.listen(3000)
```

Program-2 Write express JS script to pass data like message, name and id from express application to pug template in h1, h2 and h3 tags respectively and display data in browser.

Pug.js

```
const express = require('express');
const path = require('path');
const app = express();
app.set('view engine', 'pug');
app.set('views',__dirname);
app.get('/', (req, res) => {
res.render('example', {message: 'Hello from Express!',name: 'lju',id: 2 });
});
app.listen(6002)
```

example.pug

```
html
head
title My Express App
body
h1 #{message}
h2 #{name}
h3(style="color:red") #{id}
```

Example:3

Write express js script to load student form using pug file which contains following fields Name(text),Email(email),Course(radio : CE, IT, CSE).Once form submitted then data must be displayed on '/student' page using pug file. Means data should be submitted from express application to PUG file.

Public/student.pug

```
html
head
title Pug Form
body
    h2 Student Form
    form(action="/data",method="get")
       div
       label Enter Your Name
       input(type="text", name="name")
      div
      label Enter Your Email
      input(type="email", name="email")
      div
      label Course
      input(type="radio", name="course", value="IT",id="IT")
      |IT
      input(type="radio", name="course", value="CE",id="CE")
      |CE
      input(type="radio", name="course", value="CSE",id="CSE")
      |CSE
      br
      input(type="submit",value="Submit")
```

src/student_form.js

```
var expr = require("express");
var app = expr();
app.set("view engine","pug");
var p = require("path");
const staticpath = p.join(__dirname,"../public");
app.get("/",(req,res)=>{
res.render(staticpath+"/student");
});
app.get("/student",(req,res)=>{
res.render(staticpath+"/form_output",{name:req.query.name, email:req.query.email,
course:req.query.course});
});
app.listen(5000);
```

public/form_output.pug

```
html
head
title FORM Data
body
h1 Welcome!
table(border='1px solid',style='border-collapse:collapse;color:blue', cellpadding=10)
tr
th Name
th Email
th Course
tr
td #{name}
td #{email}
td #{course
```

File Upload

File upload is a common operation for any applications. In Node.js, with the Express web framework and the **Multer** library, adding file upload feature to your app is very easy.

What is Multer?

Multer is a node.js middleware for handling **multipart/form-data**, which is primarily used for uploading files.

What is Multipart Data?

In general, when a "form" is submitted, browsers use " *application-xx-www-form-urlencoded* " content-type. This type contains only a list of keys and values and therefore are not capable of uploading files. Whereas, when you configure your form to use **" *multipart/form-data* "** content-type, browsers will create a "multipart" message where **each part will contain a field of the form**. A multipart message will consist of text input and file input. This way using multipart/form-data you can upload files.

Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form.

Multer will not process any form which is not multipart (*multipart/form-data).*

Adding Multer

Before using multer, we have to install it using npm.

```
npm install multer
```

We will now load ***multer*** in the ***app.js*** file using the ***require()*** method.

```
const multer = require('multer');
```

DiskStorage

The disk storage engine gives you full control over storing files to disk. We will create a storage object using the ***diskStorage*** () method.

The following code will go in ***app.js*** :

```
var storage = multer.diskStorage({
   destination: 'uploads'),
   filename: function (req, file, cb) {
      cb(null , file.originalname);
   }
});
```

destination – destination is used to give folder name where file will uploade. If no destination is given, the operating system's default directory for temporary files is used.
filename - It is used to determine what the file should be named inside the folder. If you don't provide any filename, each file will be given a random name without any file extension. (here, cb is callback function). The 2 arguments to *cb* are:

- ***null*** - as we don't want to show any error.
- ***file.originalname*** - here, we have used the same name of the file as they were uploaded. You can use any name of your choice.

Other Options in Upload

1. ***limits*** - You can also put a limit on the size of the file that is being uploaded with the help of using *limits.*

   The following code will go inside the ***multer()*** .

```
// inside multer({}), file upto only 1MB can be uploaded
const upload = multer({
        storage: storage,
        limits : {fileSize : 1000000}
});
```

Here, fileSize is in bytes. (1000000 bytes = 1MB)

2. ***fileFilter*** - Set this to a function to control which files should be uploaded and which should be skipped

```
function fileFilter (req, file, cb) {
  // Allowed ext
   const filetypes = /jpeg|jpg|png|gif/;

 // Check ext
  const extname =
filetypes.test(path.extname(file.originalname).toLowerCase())
;
 // Check mime
 const mimetype = filetypes.test(file.mimetype);

 if(mimetype && extname){
     return cb(null,true);
 } else {
     cb('Error: Images Only!');
 }
}
```

Uploading Multiple Files

We can upload multiple files as well. In this case, multer gives us another function called . **arrays(fieldname[, max_count])** that accepts an array of files, all with the name ***fieldname*** . It generates an error if more than ***max_count*** files are uploaded. The array of files will be stored in req.files.

```
// uploading multiple images together
app.post("/images", upload.array("demo_images", 4), (req, res) =>{
  try {
    res.send(req.files);
  } catch (error) {
    console.log(error);
    res.send(400);
  }
```

```
});
```

Example: Write an express js script that accepts single file to be uploaded using the multer middleware and saves the file to the specific directory called "upload".

FileUpload.js

```
const expr=require("express");
const app=expr();
const path=require("path");
const multer=require("multer");
var storage=multer.diskStorage(
    {
        destination:"upload",
        filename:function(req,file,cb)
        {
            cb(null,file.fieldname+"-"+Date.now()+".jpg");
        }
    });
var upload1=multer(
    {
        storage:storage,

    }).single("mypic");
app.get("/",(req,res)=>
{
   res.sendFile(__dirname+"/index.html");
});
app.post("/uploadfile",upload1,(req,res)=>
{
     //Const f=req.file;
     //res.send(f);
       res.send("file uploaded");
    });
});
app.listen(8000,()=>
{
    console.log("server running at 8000");
});
```

Index.html

```
<form action="/uploadfile" enctype="multipart/form-data" method="post">
    <input type="file" name="mypic"/>
    <input type="submit" value="Upload"/>

</form>
```

Example: Write an express js script that accepts multiple files max number 5 to be uploaded using the multer middleware and saves the files to the specific directory called "multiple". limit the size of uploaded file to 1MB.

```
const expr=require("express");
const app=expr();
const path=require("path");
const multer=require("multer");
var storage=multer.diskStorage(
    {
        destination:"multiple",
        filename:function(req,file,cb)
        {
            cb(null,file.originalfilename);
        }
    });
const maxSize=1*1024*1024;
var upload1=multer(
    {
        storage:storage,
        limits:
        {
            fileSize:maxSize
        },

    }).array("mypic",5);
app.get("/",(req,res)=>
{
   res.sendFile(__dirname+"/index.html");
});
app.post("/uploadfile",upload1,(req,res)=>
{
     Const f=req.file;
     res.send(f);
    });
});
app.listen(8000,()=>
{
    console.log("server running at 8000");
});
```

Index.html

```
<form action="/uploadfile" enctype="multipart/form-data" method="post">
    <input type="file" name="mypic"/>
    <input type="submit" value="Upload"/>

</form>
```

Example

Write an express js script that allows only pdf type file to be uploaded using the multer middleware and saves the file to the specific directory called "specific". If file other than pdf has been uploaded then it will give an error message that "Only PDF format allowed". Also, final output to be displayed in browser through pug file.

Specific.js

```
const express = require('express')
const multer = require('multer');
const app = express();
app.set("view engine","pug");
app.get('/',(req,res)=>{
   res.sendFile(__dirname + " /index.html");
})
var store = multer.diskStorage({
   destination:"specific",
   filename: function (req, file, cb) {
                        cb(null, file.originalname)
                                }
 })
// Allow only pdf to be uploaded
const filter = (req, file, cb) => {
   if (file.mimetype == "application/pdf" ) {
        cb(null, true);
   }
   else {
         return cb('Only pdf format allowed!');
   }
}
var upload = multer({
   storage:store,
   fileFilter: filter
}).single("mypic")
app.post('/uploadfile', upload, (req, res) => {
   const file = req.file
   if (file) {
      res.render(__dirname+"/output",{file:file.originalname})
   }
})
app.listen(6789);
```

output.pug

```
html
 head
  title FORM Data
 body
```

```
h1 File #{file} have been uploaded
```

index.html

```
<html>
  <form action="/uploadfile" method="post" enctype="multipart/form-data">
    <input type="file" name="mypic" accept=".pdf"/>
    <input type="submit" value="Upload"/>
  </form>
</html>
```

RESTful API

What is REST?

The REST stands for **REpresentational State Transfer**.

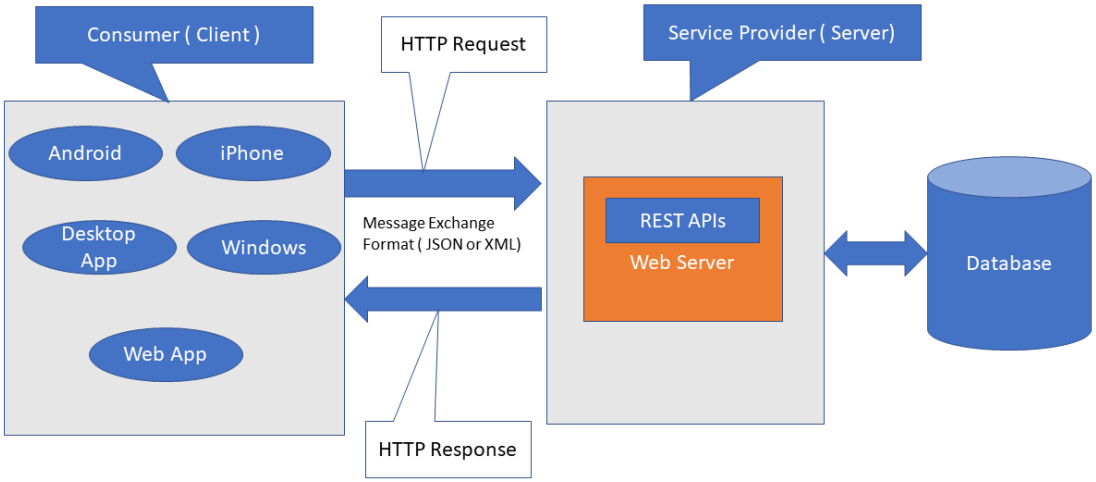
Let's understand the meaning of each word in the REST acronym.

- **State** means data
- **REpresentational** means formats (such as XML, JSON, YAML, HTML, etc)
- **Transfer** means carrying data between consumer and provider using the HTTP protocol

A REST API is an intermediary Application Programming Interface that enables two applications to communicate with each other over HTTP, much like how servers communicate to browsers.

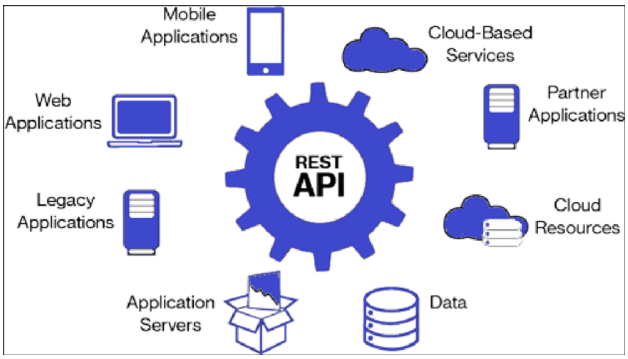
The need for REST APIs increased a lot with the drastic increase of mobile devices. It became logical to build REST APIs and let the web and mobile clients consume the API instead of developing separate applications.

REST Architecture



Request and Response: Request is the input to a web service, and the response is the output from a web service.
Message Exchange Format: It is the format of the request and response. There are two popular message exchange formats: XML and JSON.
Service Provider or Server: The service provider is one that hosts the web service.
Service Consumer or Client: A service consumer is one who is using a web service.

In simple words A RESTful API is an architectural style for an application program interface (API) that uses HTTP requests to access and use data.



What Is express.Router() For

Express Routers are a way to organize your Express application such that your primary app.js file does not become bloated and difficult to reason about

To create an instance of an Express Router, we call the **.Router()** method on the top-level Express import. Then to use that router, we mount it at a certain path using **app.use()** and pass in the router as the second argument. This router will now be used for all paths that begin with that path segment. To create a router to handle all requests beginning with **/hello**, the code would look like this

```
const helloRouter = express.Router();


app.use('/hello', helloRouter);
```

Unit-5 Express.js

EXAMPLE: IN below example **movies.js** is serve as **provider** and **index.js** is serve as **consumer**

Index1.js

```
const expr=require("express");
const app=expr();
const movies=require("./movies"); // require movies.js file
app.use("/",movies);
app.listen(3000,()=>
{
    console.log("Running at 3000");
});
```

movies.js

```
const expr=require("express");
const router1=expr.Router();
const movi=[
    {
        id:102,
        name:"abc1",
        year:1999,
        rating:8.1
    },
    {
        id:202,
        name:"xyz1",
        year:2000,
        rating:9.1
    }];
    module.exports=router1;
    router1.get("/",(req,res)=>
    {
        res.json(movi);
    });
```

To Retrieve specified element from JSON Array(API)

Index1.js

```
const expr=require("express");
const app=expr();
const movies=require("./movies"); // require movies.js file
app.use("/",movies);
app.listen(3000,()=>
{
    console.log("Running at 3000");
});
```

movies.js

```
const expr=require("express");
const router1=expr.Router();
const movi=[
    {   id:102,
        name:"abc1",
        year:1999,
        rating:8.1
    },
    {
        id:202,
        name:"xyz1",
        year:2000,
        rating:9.1
    }];
    module.exports=router1;
    router1.get("/:id1([0-9]{3,})",(req,res)=>

    {   var currMovi=movi.filter((m)=>
        {
            console.log(m)
            if(m.id==req.params.id1)
            {
                return true;
            }
        });
        console.log(currMovi)
        if(currMovi.length==1)
        {
            res.json(currMovi[0]);
        }
        else
        {
            res.json("Not Found");
        }
    });
```

NodeMailer

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm:

```
>npm install nodemailer
```

After you have downloaded the Nodemailer module, you can include the module in any application:

```
var nodemailer = require('nodemailer');
```

Example

```
var nm=require("nodemailer");
var trans=nm.createTransport(
{
    service:"gmail",
    host:"smtp.gmail.com",      //optional
    port:465,     //optional
    secure:"true",
    auth:
    {
        user:"xyz.mnc123@gmail.com",
        pass:"vcguapkksfcungnd"
    }
});
var mailOption=
{
    from:"xyz.mnc123@gmail.com",
    to:"pqr.45abc@gmail.com",
    subject:"test mail",
    text:"its easy"
}
trans.sendMail(mailOption,(err,info)=>
{
    if(err)
    {
        console.log(err);
    }
    else
    {
        console.log("email send"+info.response);
    }
});
```

What Is express.Router() For

Express Routers are a way to organize your Express application such that your primary app.js file does not become bloated and difficult to reason about

Express.Router

To create an instance of an Express Router, we call the **.Router()** method on the top-level Express import. Then to use that router, we mount it at a certain path using **app.use()** and pass in the router as the second argument. This router will now be used for all paths that begin with that path segment. To create a router to handle all requests beginning with **/hello**, the code would look like this:

```
const helloRouter = express.Router();


app.use('/hello', helloRouter);
```