ExpressJS

https://expressjs.com/
http://expressjs.com/tr/api.html
https://expressjs.com/en/guide/database-integration.html

Nodemon

https://github.com/remy/nodemon
https://www.npmjs.com/package/nodemon
Nodemon is development utility that watches/monitor npm project files for changes. If they change it will restart the script. We will use **nodemon** instead of **node** to start our script while development.

Install Nodemon as global dependency

Installing nodemon will install an **nodemon** command line utility
$ npm install nodemon -g

Use nodemon to run ES5 code

$ nodemon app.js

Use nodemon to run ES6 code

$ nodemon app.js --exec babel-node --presets es2015

Use nodemon from npm script

Add nodemon startup commands in **package.json** file

```
{
 "name": "My Express JS App",
 "version": "1.0.0",
 "main": "app.js",
 "scripts": {
   "start-es5": "nodemon app_01_es5.js",
   "start-es6": "nodemon app_02_es6.js --exec babel-node --presets es2015"
 },
 "author": "",
 "license": "ISC",
 "keywords": [],
 "description": "",
```

```
 "dependencies": {},
 "devDependencies": {
   "babel-preset-es2015": "^6.24.1"
 }
}
```

Then scripts can be started using
$ npm run start-es5
$ npm run start-es6

Install and running a simple app

Install

$ npm install express

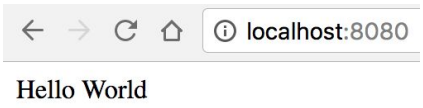
Application

```
import express from "express";

// Create application object
let app = express();

// Define a route and send response back
app.get("/", (request, response) => {
   response.send("Hello World");
});

// Start the express app by listening on a port
// Optionally write a callback that will run on startup
app.listen(8080, () => {
   console.log("Server started on port 8080");
});
```



Sending back Static file

```
import express from "express";

let app = express();

app.get("/", (request, response) => {
   response.sendfile("./app02.html");
});

app.listen(8080);
```

Sending back JSON response and Path params

http://expressjs.com/tr/api.html#res.json

The example below can accessed from below URLs.
http://localhost:8080/user-profile
http://localhost:8080/user-profile/muhammad

JSON response can be returned by sending JS Object to response.json()
Multiple resource URLs can be given in an array as first argument.
URL path params can be given using colon.

```
import express from "express";

let app = express();

app.get(["/user-profile", "/user-profile/:userName"], (request, response) => {
   let myUser = {name: "Sheraz", salary: 100};
   if (request.params.userName) {
      myUser.name = request.params.userName;
      myUser.message = "Hello " + request.params.userName;
   }
   // http://expressjs.com/tr/api.html#res.json
   response.json(myUser);
});

app.listen(8080);
```

Redirecting Request

Below example GET request on
http://localhost:8080
And then redirects to
http://localhost:8080/user-profile

```
import express from "express";

let app = express();

// Redirects to /user-profile
app.get("/", (request, response) => {
  // All http status codes
  // https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  // http://expressjs.com/tr/api.html#res.redirect
  response.redirect(301, "/user-profile");
});

app.listen(8080);
```

GET Submit Form - Read GET Parameter

http://expressjs.com/tr/api.html#req.query

```
import express from "express";
import bodyParser from "body-parser";
let app = express();

app.get("/", (request, response) => {
   response.sendFile(__dirname + "/user_register.html");
});

app.get("/users", (request, response) => {
   console.log(request.query);
   console.log("Email =", request.query.email);
   console.log("Password =", request.query.password);
   response.sendFile(__dirname + "/user_register_confirm.html");
});

app.listen(8080, () => console.log("Server Started on port 8080"));
```

POST Submit Form - Read POST Parameter - Body Parser

http://expressjs.com/tr/api.html#req.body
When a form is submitted using POST then we need a middleware to parse request body.
We can use body-parser library to parse **application/x-www-form-urlencoded**
https://www.npmjs.com/package/body-parser

If Body Parser is not configured then **request.body** is undefined.
Once Body Parser is configured then we can read POST request parameters from **request.body** Object
Tip: search "body-parser" on npmjs.com to find more examples.

```
import express from "express";
import bodyParser from "body-parser";
let app = express();

app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true })); // for parsing
application/x-www-form-urlencoded

app.get("/", (request, response) => {
   response.sendFile(__dirname + "/user_register.html");
});

app.post("/", (request, response) => {
   console.log(request.body);
   console.log("Email =", request.body.email);
   console.log("Password =", request.body.password);
   response.sendFile(__dirname + "/user_register_confirm.html");
});

app.listen(8080, () => console.log("Server Started on port 8080"));
```

Templating - view engine

https://expressjs.com/en/guide/using-template-engines.html
Templating is used to modify view HTML on the server before sending it to client.

List of all template engine:

https://github.com/expressjs/express/wiki#template-engines

Steps to use templates

In the example below we will configure EJS template view engine.
https://www.npmjs.com/package/ejs

1. Set view engine

```
let app = express();
app.set("view engine", "ejs");
```

2. Page data and render view

We can optionally pass data to the view.
Call response.render() to render view.

```
let pageData = {
    pageTitle: "Create User"
};
response.render("./pages/user_register", pageData);
```

3. Use Page data in the view

```
<title>User Register - <%= pageTitle %></title>
```

Serving static files

app.use(express.static('images'));

Session

https://www.npmjs.com/package/express-session

Express Generator

https://www.npmjs.com/package/express-generator
Express Generator is a utility to quickly generate express application.

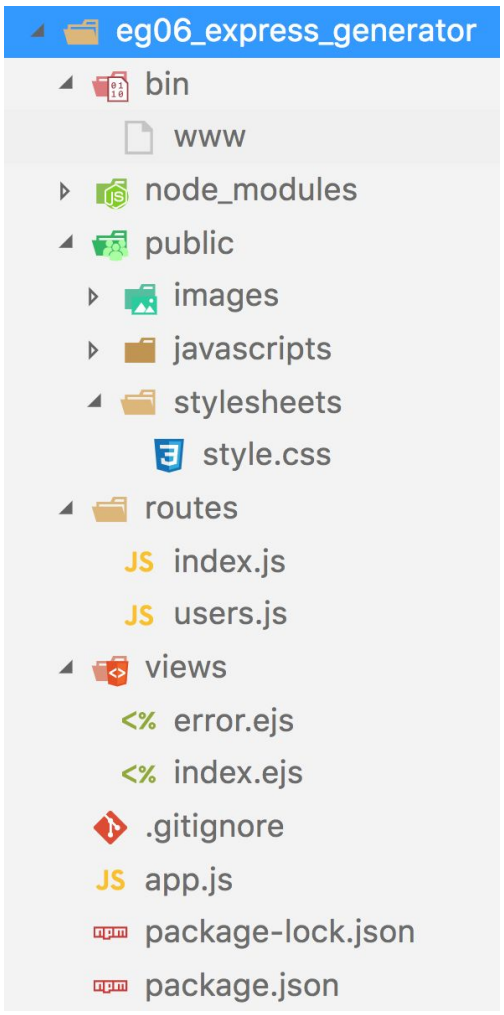
Install global express-generator command

$ npm install express-generator -g

Create application

$ express --view=ejs
$ npm install

Project file structure



bin/www

- application startup script.
- Sets up http server and its port

public

- Contains all static assets like images, client side javascript, and stylesheets
- If our client side application is created in frameworks like ReactJS, Angular JS or Angular then this folder will contain all of those files.

routes

- This is where all of our route code will reside.
- Or in other words our MVC application controller

views

- Contains all our MVC application views
- These views are created in our choice of view engine/template

app.js

- Sets up express app engine
- Sets up middlewares like body-parser, cookie-parser, logging
- Initialize routes
- Sets up view folder
- Sets up static assets folder
- Sets up error pages like 404 and 500

NOTE: most of our coding work will be in **routes** and **views**.

Run Application

To run the application we can use any of these methods

npm script

$ npm start

node

$ node bin/www

IDE

Execute bin/www script using IntelliJ or VS Code

Testing Application

By default express generator create 2 example pages. Visit them to verify if application has started successfully
http://localhost:3000/
http://localhost:3000/users