Vocabulary

Scaffolding -- the technique of generating a generic application with features common to most modern applications already built-in (Also called “boilerplate”).
Model-View-Controller (MVC) -- The MVC pattern is a way of organizing software for building user interfaces and websites. It breaks down the program into three parts: the model (dealing with information), the view (displaying information to the user), and the controller (connecting the model and the view).

Scaffolding

Recreating identical code repeatedly can be a tedious task, making the use of 'boilerplate' code an essential strategy for simplifying the process. Express acknowledges this need by providing a utility for generating scaffolding to initiate an Express project. Despite its utility, it falls short in constructing the most optimal framework, lacking support for certain libraries such as Handlebars.

The Veilmoor Travel Website

Will be a running example; A fictional website for a fictional town called “Veilmoor.” This project will be making use of many techniques including REST application. Try to keep your web application files separated from the other files by for example ‘***~/projects/veilmoor/***’.

npm manages project dependencies --as well as metadata about the project -- in a file called *package.json*. Everytime you run npm, you’ll get warnings unless you provide a repository URL in *package.json*, and a nonempty ***README.md*** file.

Unless you’re using a hosting service or deployment system that requires your main application file to have a specific name there is no reason for using naming conventions like naming your main file app.js. `npm init` will default the main file to ***index.js***.

We also replaced Node’s low-level `res.end` with `res.send` instead and Node’s `res.writeHead` with `res.status`. We are free of using `res.writeHead` and `res.end` but it’s not recommended. We are going to be using `app.use` instead of `app.get` because we want Express to add middleware. We need to be cognizant that in Express, the order in which routes and middleware are added is significant. If we put the 404 handler above the routes, the home page and About page would stop working resulting in a 404 as we try to visit any page.

We may encounter issues when using wildcards in `res.get`. For example, in the following example:

```
        app.get(‘/about*’, function(req, res) {
```

```
        // Send content…
    });

    app.get('/about/contact', function(req, res){
        // Send content…
    });
    app.get('/about/directions', function(req, res) {
        // Send content…
    });
```

In this example, the */about/contact* and */about/directions* handlers will never be matched because the first handler uses a wildcard in its path: */about**.

To make vim recognize *.handlebars* files add the line `au BufNewFile,BufRead *.handlebars set file type=html` to *~/.vimrc* file. Vim will not create a *.vimrc* file for us automatically, you will need to create it ourselves.

```
:!mkdir $HOME/vimfiles
:e $HOME/vimfiles/vimrc
```

Initial Steps

1) Create a new directory for your project. We are going to refer to this directory as the "project directory," "app directory," or "project root."
2) In your terminal, run: `npm init` and answer the following questions the best you can. (For the "entry point" question, use the name of the main file of your project, in our case, ***veilmoor.js***)
3) We install Express by running the following command: `npm install --save express`
4) We create a ***.gitignore*** file to make sure we don't accidentally push node_module to our github. Inside the ***.gitignore*** file we will write the following.

   Source: *"root/.gitignore"*

   ```
       # ignore packages installed by npm
       node_modules

       # put any other files you don't want to check in here,
       # such as .DS_Store (OSX), *.bak, etc.
   ```

5) Now we can create the file ***veilmoor.js*** file. This will be our project entry point. If ***helloWorld.js*** is in the project you may delete it or put it somewhere else. We will be referring to ***veilmoor.js*** as our "app file."

In ***veilmoor.js*** add the following code:

Source: *"root/veilmoor.js"*

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

var app = express();

// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set('port', process.env.PORT || 3000);

// Custom 404 (Not Found) page
// The following function will be executed
// for every request to the app
app.use(function(req, res) {

      res.type('text/plain');
      res.status(404)
      res.send('404 - Not Found');

});

// Custom 500 (Internal Server Error) page
app.use(function(err, req, res, next) {
      console.error(err.stack);
      res.type('text/plain');
      res.status(500);
      res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
      console.log( 'Express started on http://localhost:' +
            app.get('port') + '; press Ctrl-C to terminate.');
});
```

6) Test your code by running `node veilmoor.js`.
7) Navigate to http://localhost:3000.
8) Let's add some routes for the home page and the about page.
   Source: *"root/veilmoor.js"*

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

var app = express();




// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set('port', process.env.PORT || 3000);

// NOTE: This is a different app.get than what we used before
// Click the link in app.get to see the main differences.

// Both functions return plaintext with a status code of 200 (OK)
// 200 is the default status code and
// doesn't have to be specified.
app.get('/', function(req, res) {
      res.type('text/plain');
      res.send('Veilmoor Travel');
});

app.get('/about', function(req, res) {
      res.type('text/plain');
      res.send('About Veilmoor Travel');
});

// Custom 404 (Not Found) page
// The following function will be executed
// for every request to the app
app.use(function(req, res) {

      res.type('text/plain');
      res.status(404)
      res.send('404 - Not Found');

});

// Custom 500 (Internal Server Error) page
app.use(function(err, req, res, next) {
      console.error(err.stack);
```

```
        res.type('text/plain');
        res.status(500);
        res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
        console.log( 'Express started on http://localhost:' +
                app.get('port') + '; press Ctrl-C to terminate.');
});
```

9) Run npm veilmoor.js
10) Test your new website by going to the following links: http://localhost:3000, http://localhost:3000/about, http://localhost:3000/sdjalsdjaslk. If for some reason we get a 500 error, double check the file *veilmoor.js* for all the code is correct.

Views and Layouts

A View in the model-view-controller paradigm is what is delivered to the user to see. This does not mean the view is a static asset like a css file or an image. The HTML can be constructed on the fly to prove a customized page for each request. For this project we will be using Eric Ferraiuolo's express3-handlebars package. Layouts free you from having to update thousands of static html pages everytime there is one little change in code they have in common.

Initial Steps:

1) Execute: `npm install --save express3-handlebars`
2) Add the following lines to *veilmoor.js*

   Source: *"root/veilmoor.js"*

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

var app = express();

// Set up handlebars view engine
// Creates a view engine and configures
// Express to use it by default.
var handlebars = require('express3-handlebars')
```

```
// defaultLayout: ‘main’ means that, unless we specify otherwise,
// this is the layout that will be used for any view.
      .create({ defaultLayout: ‘main’ });
app.engine(‘handlebars’, handlebars.engine);
app.set(‘view engine’, ‘handlebars’);

// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set(‘port’, process.env.PORT || 3000);


// NOTE: This is a different app.get than what we used before
// Click the link in app.get to see the main differences.

// Both functions return plaintext with a status code of 200 (OK)
// 200 is the default status code and
// doesn’t have to be specified.
app.get(‘/’, function(req, res) {
      res.type(‘text/plain’);
      res.send(‘Veilmoor Travel’);
});

app.get(‘/about’, function(req, res) {
      res.type(‘text/plain’);
      res.send(‘About Veilmoor Travel’);
});


// Custom 404 (Not Found) page
// The following function will be executed
// for every request to the app
app.use(function(req, res) {

      res.type(‘text/plain’);
      res.status(404)
      res.send(‘404 - Not Found’);

});

// Custom 500 (Internal Server Error) page
app.use(function(err, req, res, next) {
      console.error(err.stack);
      res.type(‘text/plain’);
```

```
        res.status(500);
        res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
        console.log( 'Express started on http://localhost:' +
                app.get('port') + '; press Ctrl-C to terminate.');
});
```

3) Create a directory called ***views*** that has a subdirectory called ***layouts***.

```
Parent dir
        |_ views
                |_ layouts
        |_public
                |_img
                        |_logo.jpg
                |_home.html
                |_about.html
                |_notfound.html
```

4) Create a file called *main.handlebars* inside of *views/layouts*

```
Parent dir
        |_ views
                |_ layouts
                        |_ main.handlebars
        |_public
                |_img
                        |_logo.jpg
                |_home.html
                |_about.html
                |_notfound.html
```

5) Add the following to *main.handlebars*:

Source: *root/views/layouts/main.handlebars*

```
<!doctype html>
<html>
<head>
        <title>Veilmoor Travel </title>
</head>
```

```
<body>
      <!- Replaced with the HTML for each view -->
      {{{ body }}}
</body>
</html>
```

6) Then, let's create *home.handlebars*,…
   Source: *root/views/home.handlebars*

```
<h1> Welcome to Veilmoor Travel </h1>
```

7) … *about.handlebars*, …
   Source: *root/views/about.handlebars*

```
<h1> About Veilmoor Travel </h1>
```

8) … *404.handlebars*, …
   Source: *root/views/404.handlebars*

```
<h1> 404 - Not Found </h1>
```

9) … and *500.handlebars*
   Source: root/views/500.handlebars

```
<h1> 500 - Server Error </h1>
```

10) Replace our old routes with new routes that use these views:

Source: *"root/veilmoor.js"*

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

var app = express();

// Set up handlebars view engine
// Creates a view engine and configures
// Express to use it by default.
var handlebars = require('express3-handlebars')
// defaultLayout: 'main' means that, unless we specify otherwise,
// this is the layout that will be used for any view.
      .create({ defaultLayout: 'main' });
app.engine('handlebars', handlebars.engine);
app.set('view engine', 'handlebars');

// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set('port', process.env.PORT || 3000);


// NOTE: This is a different app.get than what we used before
// Click the link in app.get to see the main differences.

// Both functions return a status code of 200 (OK) by default
app.get('/', function(req, res) {
      res.render('home');
});

app.get('/about', function(req, res) {
      res.render('about');
});


// Catch-all 404 (Not Found) handler (middleware)
// The following function will be executed
// for every request to the app
app.use(function(req, res) {

      res.type('text/plain');
      res.status(404)
      res.send('404 - Not Found');

});

// 500 (Internal Server Error) handler (middleware)
app.use(function(err, req, res, next) {
      console.error(err.stack);
      res.type('text/plain');
      res.status(500);
      res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
      console.log( 'Express started on http://localhost:' +
            app.get('port') + '; press Ctrl-C to terminate.');
});
```

11) Start your server and check http://localhost:3000 and http://localhost:3000/about.
12) Notice a new file *root/views/layouts/main.handlebars* has appeared.

Static files and Views

Middleware as a concept provides modularization, making it easier to handle requests. The static middleware allows us to designate one or more directories as containing static resources that are simply to be delivered to the client without any special handling. This is where you would put things like images, CSS files, and client-side JavaScript files.

Steps:

1) Before the routes lets us add the static middleware:
   Source: *"root/veilmoor.js"*

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

var app = express();

// Set up handlebars view engine
// Creates a view engine and configures
// Express to use it by default.
var handlebars = require('express3-handlebars')
// defaultLayout: 'main' means that, unless we specify otherwise,
// this is the layout that will be used for any view.
      .create({ defaultLayout: 'main' });
app.engine('handlebars', handlebars.engine);
app.set('view engine', 'handlebars');

// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set('port', process.env.PORT || 3000);

// Designates root/public as the static directory
// creates a route for each static file we want to
// deliver that renders a file and returns it to the client
app.use(express.static(__dirname + '/public'));

// NOTE: This is a different app.get than what we used before
// Click the link in app.get to see the main differences.

// Both functions return a status code of 200 (OK) by default
```

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

app.get('/about', function(req, res) {
      res.render('about');
});


// Catch-all 404 (Not Found) handler (middleware)
// The following function will be executed
// for every request to the app
app.use(function(req, res) {

      res.type('text/plain');
      res.status(404)
      res.send('404 - Not Found');

});

// 500 (Internal Server Error) handler (middleware)
app.use(function(err, req, res, next) {
      console.error(err.stack);
      res.type('text/plain');
      res.status(500);
      res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
      console.log( 'Express started on http://localhost:' +
            app.get('port') + '; press Ctrl-C to terminate.');
});
```

2) Let's modify our layout so that the logo appears in every page:
   Source: *root/views/layouts/main.handlebars*

```
<!doctype html>
<html>
<head>
      <title>Veilmoor Travel </title>
```

```
        </head>
        <body>
              <!- The header element was introduced to provide -->
              <!- semantic information about content that appears -->
              <!- At the top of the page s.a. logos, title text, or
navs-->

              <header>
                    <img src=”/img/logo.jpg” alt=”Veilmoor Travel Logo”>
              </header>
              <!- Replaced with the HTML for each view -->
              {{{ body }}}
        </body>
        </html>
```

Dynamic Content in Views

Views can contain dynamic information.

Steps:

1) Let’s change *veilmoor.js* to deliver a “virtual fortune.” to the about page.
   Source: *“root/veilmoor.js”*

```
var express = require(‘express’);

var app = express();

// Set up handlebars view engine
// Creates a view engine and configures
// Express to use it by default.
var handlebars = require(‘express3-handlebars’)
// defaultLayout: ‘main’ means that, unless we specify otherwise,
// this is the layout that will be used for any view.
      .create({ defaultLayout: ‘main’ });
app.engine(‘handlebars’, handlebars.engine);
app.set(‘view engine’, ‘handlebars’);

// The port will either be specified in a .env file or
// will be set to 3000 by default
app.set(‘port’, process.env.PORT || 3000);

// Designates root/public as the static directory
```

```
      // creates a route for each static file we want to
      // deliver that renders a file and returns it to the client
      app.use(express.static(__dirname + '/public'));

      var fortunes = [
            "In the corner of your eyes you see an old woman staring at
you",
            "Your body is heavy. You feel like killing",
            "The mark burns your skin. You lose a point of health",
            "You hear a voice next to your ear. It tells you to visit
the cemetery."
      ];

      // NOTE: This is a different app.get than what we used before
      // Click the link in app.get to see the main differences.

      // Both functions return a status code of 200 (OK) by default
      app.get('/', function(req, res) {
            res.render('home');
      });

      app.get('/about', function(req, res) {
            var randomFortune = fortunes[
                  Math.floor(Math.random() * fortunes.length)
            ];
            res.render('about', { fortune: randomFortune } );
      });


      // Catch-all 404 (Not Found) handler (middleware)
      // The following function will be executed
      // for every request to the app
      app.use(function(req, res) {

            res.type('text/plain');
            res.status(404)
            res.send('404 - Not Found');

      });

      // 500 (Internal Server Error) handler (middleware)
      app.use(function(err, req, res, next) {
            console.error(err.stack);
```

```
        res.type('text/plain');
        res.status(500);
        res.send('500 - Server Error');
});

// Listen to the connections in the port -> 3000
// port is 3000 because there isn't a .env
app.listen(app.get('port'), function(){
      console.log( 'Express started on http://localhost:' +
            app.get('port') + '; press Ctrl-C to terminate.');
});
```

2) Modify the view as well
   Source: *root/views/about.handlebars*

```
<h1> About Veilmoor Travel </h1>
<p> Something happened </p>
<blockquote>{{fortune}}</blockquote>
```

3) Restart the server and load http://localhost:3000/about page. You will see a random fortune.

Code Nugget

`npm init` initialize a node project and add to the directory a *package.json* which we will be using to download node packages.
`--save` It updates the package.json after installing it in the node_module directory.

`app.set(name, value)` Assigns setting name to value. You may store any value that you want, but certain names can be used to configure the behavior of the server.
Parameters: A name and a value. Value can be retrieved by `app.get(name)`.
Return Value: Nothing

`app.use([path,] callback [, callback...])` Mounts the specified middleware function or functions at the specified path: the middleware function is executed when the base of the requested path matches path. A route will match any path that follows its path immediately with a "/". For example: `app.use('/apple', ...)` will match "***/apple***", "***/apple/images***", "***/apple/images/news***", and so on.

Since path defaults to “/”, middleware mounted without a path will be executed for every request to the app.

Parameters:

★ path (**default:** *'/' (root path)*): The path for which the middleware function is invoked; can be any of:
  - A string representing a path.
  - A path pattern.
  - A regular expression pattern to match paths.
  - An array of combinations of any of the above.

★ Callback (**default:** None): Callback functions; can be:
  - A middleware function.
  - A series of middleware functions (separated by commas).
  - An array of middleware functions.
  - A combination of all of the above.

    You can provide multiple callback functions that behave just like middleware, except that these callbacks can invoke next('route') to bypass the remaining route callback(s). You can use this mechanism to impose pre-conditions on a route, then pass control to subsequent routes if there is no reason to proceed with the current route.

Return Value: Nothing

`res.type(type)` Sets the Content-Type HTTP header to the MIME type as determined by the specified type. If type contains the “/” character, then it sets the Content-Type to the exact value of type, otherwise it is assumed to be a file extension and the MIME type is looked up in a mapping using the express.static.mime.lookup() method.

Parameters: The type parameter describes the MIME type.

Return Value: It returns an Object.

`res.status(code)` This function sets the HTTP status for the response. It works like a shortcut for Node's response.statusCode and can be used in a chain.

Parameters: A single parameter representing an HTTP status code.

Return Value: It returns an Object.

`res.send( [body] )` sends the HTTP response. The body parameter can be a String or a Buffer object or an object or an Array.

Parameters: This function accepts a single parameter body that describes the body which is to be sent in the response.

Return Value: It returns an Object.

`app.listen([port[, host[, backlog]]][, callback])` is used to bind and listen to the connections on the specified host and port. This method is identical to Node’s http.Server.listen() method.

Parameters:

★ **Port**: It specifies the port on which we want our app to listen. If the port number is omitted or is 0, the operating system will assign an arbitrary unused port, which is useful for cases like automated tasks.

★ **Host** (Optional): It specifies the IP Address of the host on which we want our app to listen. You can specify the host if and only if you have already specified the port. (since you have a closing(‘]’) bracket after ([, host[, backlog]]) as you can see in the above syntax, so this means the port must be specified before specifying host and backlog).

★ **Backlog** (Optional): It specifies the max length of the queue of pending connections. You can specify the backlog if and only if you have already specified the port and host. (since you have a closing bracket after ([, backlog]), so this means you will have to specify the host before specifying backlogs)

★ **Callback** (Optional): It specifies a function that will get executed, once your app starts listening to the specified port. You can specify callback alone i.e., without specifying port, host, and backlogs.(since this is a separate set of arguments in opening and closing brackets([, callback]), this means you can specify these arguments without specifying the argument of previous opening and closing square brackets.

Return Value: Nothing

`app.get(name)` function returns the value name app setting. The app.set() function is used to assign the setting name to value. This function is used to get the values that are assigned.

Parameters: A name referencing an item set by `app.set(name, value)`.

Return Value: It returns the Object ‘value’ mapped to ‘name.’

`app.get( path, callback )` routes the HTTP GET Requests to the path which is being specified with the specified callback functions. Basically, it is intended for binding the middleware to your application. This is the method by which we’re adding routes.

Parameters:

★ **path:** The path is what defines the route. By default, it doesn’t care about the case or trailing slash, and it doesn’t consider the querystring when performing the match. To it: ‘/about,’ ‘/About,’ ‘/about/,’ ‘/about?foo=bar,’ ‘/about/?foo=bar,’ etc… are considered the same.
★ **callback:** They can be a middleware function or a series/array of middleware functions. The function you provide will get invoked when the route is matched. The parameters passed to that function are the request and response objects.

Return Value: None

In A Nutshell

❖ Employing 'boilerplate' code is crucial to simplify the repetitive process of recreating identical code. Express recognizes this need and offers a utility for generating scaffolding to kickstart an Express project. However, while useful, Express falls short in creating the most optimal framework, particularly lacking support for certain libraries like Handlebars.

❖ This project serves as a practical illustration for a fictional website named "Veilmoor." Utilizing various techniques, including REST applications, the project organizes web application files separately, typically in the '~/projects/veilmoor/' directory.

❖ npm manages dependencies and project metadata in 'package.json,' requiring a repository URL and a nonempty README.md file to avoid warnings.

❖ Naming conventions for the main application file are flexible, defaulting to 'index.js' with npm init.

- Notable changes include replacing Node's res.end with res.send and res.writeHead with res.status.
- The use of app.use instead of app.get is recommended for adding middleware in Express.
- Care must be taken with the order of adding routes and middleware in Express, as it influences functionality. Additionally, caution is advised when using wildcards in routes, as demonstrated by potential issues in the provided example.