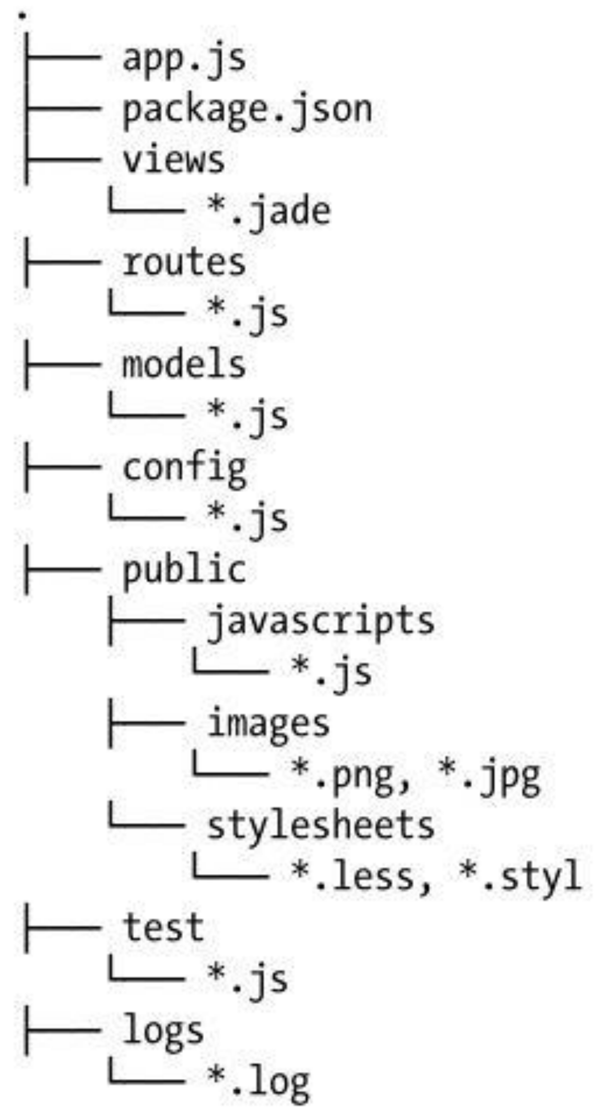
Express

Template Engine, MVC, Request&Response Object

Express application generator

- Use the application generator tool, express-generator, to quickly create an application skeleton.

```
$ npm install -g express-generator

$ express -e -c less -f MyApp
// -e ejs
// -c set the stylesheet engine to less
// -f force on non-empty directory
// -h for help

$ cd MyApp
$ npm install
```



```
.
├── app.js
├── package.json
├── views
│   └── *.jade
├── routes
│   └── *.js
├── models
│   └── *.js
├── config
│   └── *.js
├── public
│   ├── javascripts
│   │   └── *.js
│   ├── images
│   │   └── *.png, *.jpg
│   └── stylesheets
│       └── *.less, *.styl
├── test
│   └── *.js
├── logs
│   └── *.log
```

Watching for File Changes

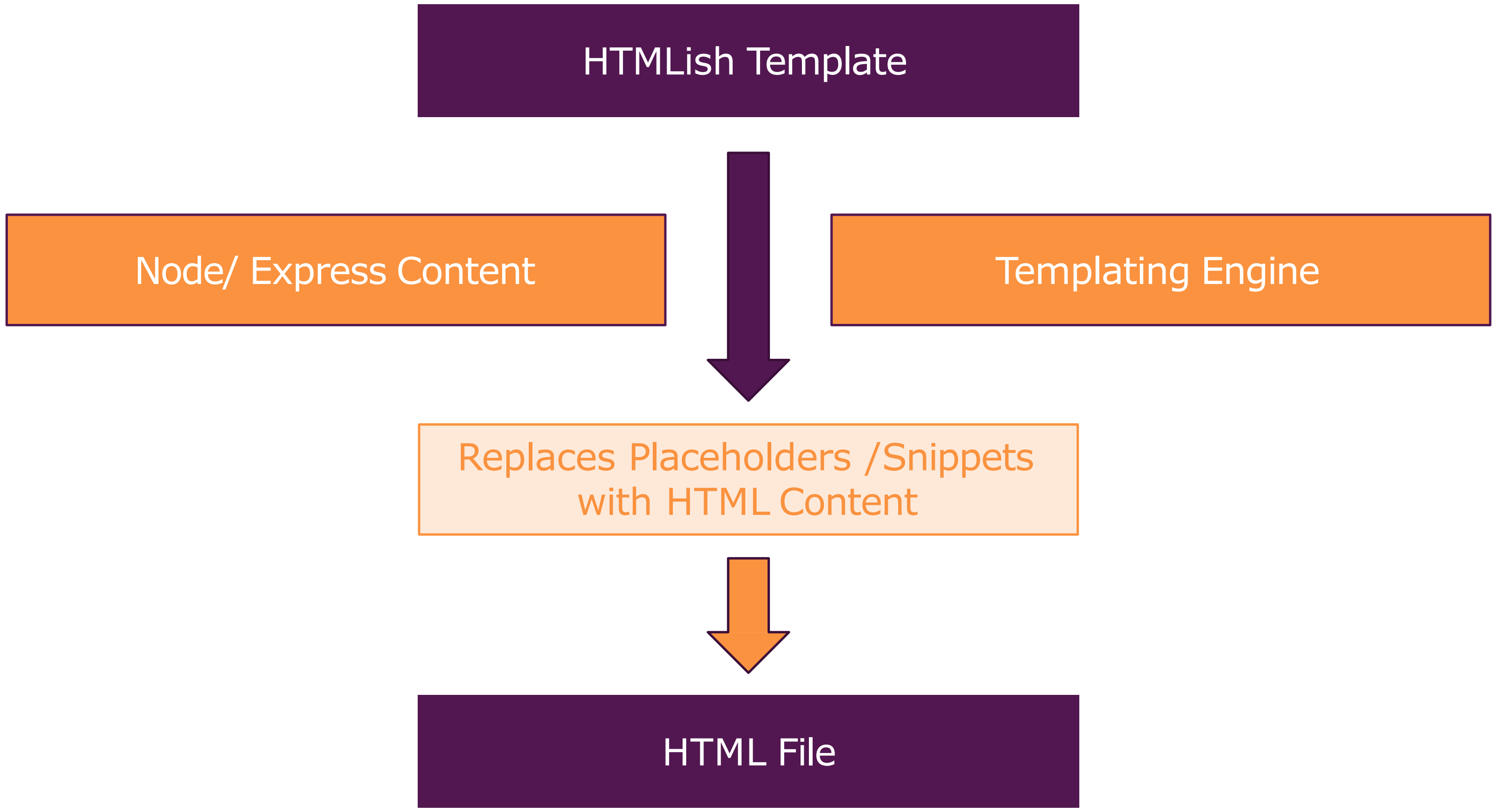
- The following file-watching tools can leverage the watch() method from the core Node.js fs module and restart our servers when we save changes from an editor.
  - **forever** https://npmjs.org/package/forever
  - **node-dev** https://npmjs.org/package/node-dev
  - **nodemon** https://npmjs.org/package/nodemon
  - **supervisor** https://npmjs.org/package *Written by the creators of NPM*
  - **up** https://npmjs.org/package/up *Written by the Express.js team*

Template Engines

- Template engines are libraries that allow us to use different template languages (EJS, Pug..)

- Template language is a special set of instructions that instructs the engine how to process data. The language is specific to a particular template engine. The instructions in the template are usually used to present data in a better format suitable for end-users.
- The process of combining data with templates is called rendering. Some template engines have functionality to compile templates as an extra step before rendering.

Template Engines



Common Template Engines

- **Jade (Pug)** allows any JavaScript in its code. uses python/haml-like syntax, which takes into account whitespace and tabs.
  - https://pugjs.org
  - https://github.com/pugjs/pug
- **Embedded JavaScript (EJS)** is another popular choice for Node.js apps and it might be a better alternative when performance is important because in benchmark tests EJS performs better than Jade.
  - https://github.com/tj/ejs
- **Handlebars:** It uses a template and an input object to generate HTML or other text formats. Handlebars templates look like regular text with embedded Handlebars expressions.
  - https://handlebarsjs.com/

Common Template Engines

EJS	Pug (Jade)	Handlebars
`<p><%= name %></p>`	`p #{name}`	`<p>{{ name }}</p>`
Use normal HTML and plain JavaScript in your templates	Use minimal HTML and custom template language	Use normal HTML and custom template language

Use a Template Engine

- Install Template Engines
  - `npm install ejs pug express-handlebars –save`

- Specify a template file/engine **extension**
  - With the `view engine` setting
    - `app.set('view engine', 'ejs’);`
    - `app.set('view engine', 'pug’);`
  - With the `render()` function
    - `response.render('index.ejs’);`
    - `response.render('index.jade’);`

- Specify the **path to your views**
  - With the `views` setting
    - `app.set('views', path.join(__dirname, 'templates’));`

HTML -> EJS

add-product.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Add Product</title>
    <link rel="stylesheet" href="/css/main.css">
</head>

<body>
    <header class="main-header">
        <nav class="main-header__nav">
            <ul class="main-header__item-list">
                <li class="main-header__item"><a href="/">Shop</a></li>
                <li class="main-header__item"><a class="active" href="/admin/add-
product">Add Product</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <form class="product-form" action="/admin/add-product" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title">
            </div>

            <button class="btn" type="submit">Add Product</button>
```

add-product.hbs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title><%= pageTitle %></title>
    <link rel="stylesheet" href="/css/main.css">
</head>

<body>
    <header class="main-header">
        <nav class="main-header__nav">
            <ul class="main-header__item-list">
                <li class="main-header__item"><a href="/">Shop</a></li>
                <li class="main-header__item"><a class="active" href="/admin/add-
product">Add Product</a></li>
            </ul>
        </nav>
    </header>

    <main>
        <form class="product-form" action="/admin/add-product" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title">
            </div>

            <button class="btn" type="submit">Add Product</button>
```

EJS Layout

head.ejs

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title><%=pageTitle%></title>
    <link rel="stylesheet" href="/css/main.css">
```

navigation.ejs

```
<header class="main-header">
    <nav class="main-header__nav">
        <ul class="main-header__item-list">
            <li class="main-
header__item"><a class="<%= path === '/' ? 'active' : '' %>" href="/
">Shop</a></li>
            <li class="main-
header__item"><a class="<%= path === '/admin/add-
product' ? 'active' : '' %>"
                    href="/admin/add-product">Add Product</a></li>
        </ul>
    </nav>
</header>
```

navigation.ejs

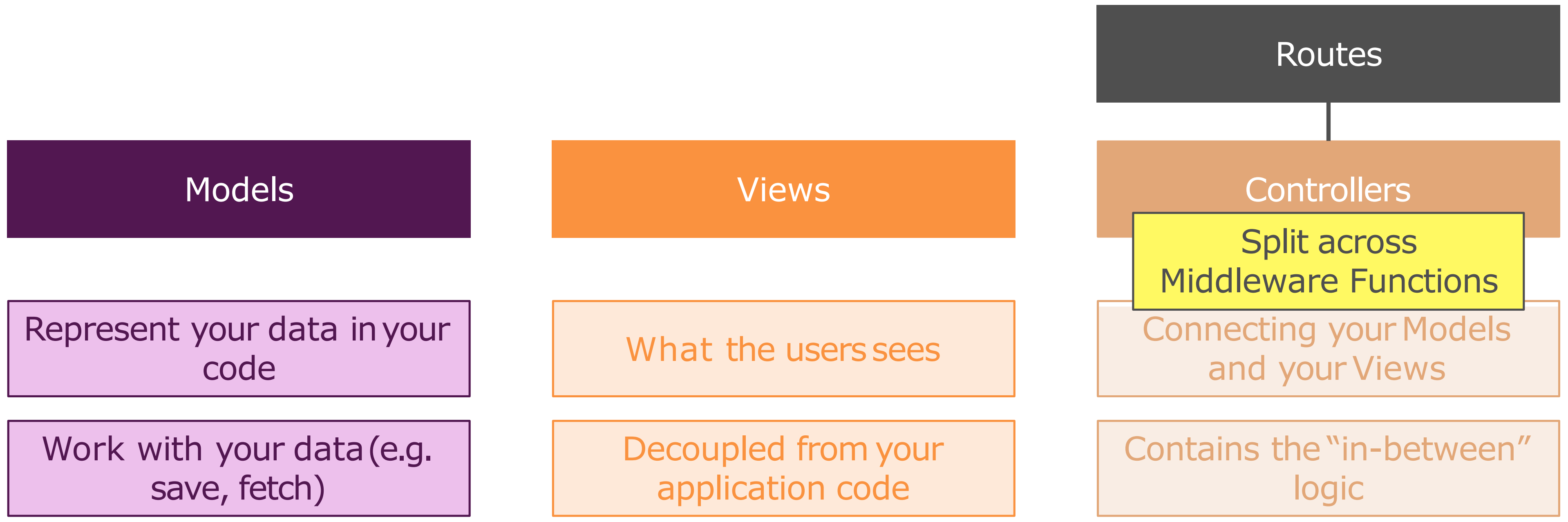
```
</body>

</html>
```

add-product.ejs

```
<%- include('fragments/head.ejs')%>
<link rel="stylesheet" href="/css/forms.css">
<link rel="stylesheet" href="/css/product.css">
</head>

<body>
    <%- include('fragments/navigation.ejs')%>

    <main>
        <form class="product-form" action="/admin/add-
product" method="POST">
            <div class="form-control">
                <label for="title">Title</label>
                <input type="text" name="title" id="title">
            </div>

            <button class="btn" type="submit">Add Product</button>
        </form>
    </main>
    <%- include('fragments/end.ejs')%>
```

What’s MVC?

Separation of Concerns


Routes
Models
Views
Controllers
Split across Middleware Functions
Represent your data in your code
What the users sees
Connecting your Models and your Views
Work with your data (e.g. save, fetch)
Decoupled from your application code
Contains the “in-between” logic

Adding a new Model

```
const products = [];

module.exports = class Product {

    constructor(title) {
        this.title = title;
    }

    save() {
        products.push(this);
    }

    static getAll() {
        return products;
    }

}
```

- Here we save our products into an in-memory array. We'll save to DB later when we introduce MongoDB.

Adding a new Controller

```
const Product = require('../models/product');

exports.getAddProduct = (req, res, next) => {
    res.render('add-product', { pageTitle: 'Add Product', path: '/admin/add-
product', formsCSS: true, productCSS: true, activeAddProduct: true });
};

exports.saveProduct = (req, res, next) => {
    // products.push({ title: req.body.title });
    const prod = new Product(req.body.title);
    prod.save();
    res.redirect('/');
};

exports.getAllProduct = (req, res, next) => {
    res.render('shop', { prods: Product.getAll(), pageTitle: 'Shop', path: '/', formsCSS: true, p
roductCSS: true, activeShop: true });
};
```

Modify our Router

```
const express = require('express');
const productController = require('../controllers/products');

const options = {
    "caseSensitive": false,
    "strict": false
};
const router = express.Router(options);


router.get('/add-product', productController.getAddProduct);

router.post('/add-product', productController.saveProduct);

module.exports = router;
```

Exercise: Add Product functionality

```
exports.postAddProduct = (req, res, next) => {
    const title = req.body.title;
    const imageUrl = req.body.imageUrl;
    const price = req.body.price;
    const description = req.body.description;
    const product = new Product(title, imageUrl, price, description);
    product.save();
    res.redirect('/');
};
```

```
<form class="product-form" action="/admin/add-product" method="POST">
 <div class="form-control">
     <label for="title">Title</label>
     <input type="text" name="title" id="title">
 </div>
 <div class="form-control">
     <label for="imageUrl">Image URL</label>
     <input type="text" name="imageUrl" id="imageUrl">
 </div>
 <div class="form-control">
     <label for="price">Price</label>
     <input type="number" name="price" id="price" step="0.01">
 </div>
 <div class="form-control">
     <label for="description">Description</label>
     <textarea name="description" id="description"></textarea>
 </div>

 <button class="btn" type="submit">Add Product</button>
</form>
```

```
<div class="card__image">
 <img src="<%= product.imageUrl %>" alt="A Book">
</div>
<div class="card__content">
 <h2 class="product__price"><%= product.price %></h2>
 <p class="product__description"><%= product.description %></p>
</div>
```

Exercise: Display Product Detail functionality

shop/product-list.ejs

```
<div class="card__actions">
  <a href="/products/<%= product.id %>" class="btn">view Detail</a>
  <button class="btn">Add to Cart</button>
</div>
```

models/product.js

```
save() {
    this.id = Math.random().toString();
    products.push(this);
}

static findById(prodId) {
    return products.find(p => p.id === prodId);
}
```

controllers/shop.js

```
exports.getProduct = (req, res, next) => {
    const prodId = req.params.prodId;
       res.render('shop/product-detail', {
        product: Product.findById(prodId),
        pageTitle: 'Product Detail',
        path: '/products',
    })
}
```

Exercise: Edit Product Detail functionality

routes/admin.js

```
router.get('/edit-product/:prodId', adminController.getEditProduct);
router.post('/edit-product', adminController.postEditProduct);
```

controllers/admin.js

```
exports.getEditProduct = (req, res, next) => {
    const prodId = req.params.prodId;
    res.render('admin/edit-product', {
        product: Product.findById(prodId),
        pageTitle: 'Edit Product',
        path: '/admin/products',
    })
}
exports.postEditProduct = (req, res, next) => {
    const id = req.body.id; const title = req.body.title;
    const imageUrl = req.body.imageUrl; const price = req.body.price;
    const description = req.body.description;
    const product = new Product(id, title, imageUrl, price, description);
    product.update();
    res.redirect('/admin/products');
}
```

Exercise: Edit Product Detail functionality

views/admin/products.ejs

```
<div class="card__actions">
        <a href="/admin/edit-product/<%= product.id %>" class="btn">Edit</a>
</div>
```

views/admin/edit-product.ejs

```
<form class="product-form" action="/admin/edit-product" method="POST">
        <input type="text" name="id" id="id" value="<%= product.id %>" readonly>
        <input type="text" name="title" id="title" value="<%= product.title %>">
        <input type="text" name="imageUrl" id="imageUrl" value="<%= product.imageUrl %>">
        <input type="number" name="price" id="price" step="0.01" value="<%= product.price %>">
        <textarea name="description" id="description"><%= product.description %></textarea>

        <button class="btn" type="submit">Edit Product</button>
</form>
```

Exercise: Delete Product functionality

views/admin/products.ejs

```
<form action="/admin/delete-product" method="POST">
        <input type="hidden" name="id" value="<%= product.id %>">
        <button class="btn" type="submit">Delete</button>
</form>
```

routes/admin.js

```
router.post('/delete-product', adminController.postDeleteProduct);
```

controllers/admin.js

```
exports.postDeleteProduct = (req, res, next) => {
    Product.deleteById(req.body.id);
    res.redirect('/admin/products');
}
```

models/product.js

```
static deleteById(prodId) {
   products = products.filter(p => p.id !== prodId);
}
```

MVC Summary

Model

- Responsible for representing your data
- Responsible for managing your data (saving, fetching, …)
- Doesn’t matter if you manage data in memory, files, databases
- Contains data-related logic

View

- What the user sees
- Shouldn’t contain too much logic (Handlebars!)

Controller

- Connects Model and View
- Should only make sure that the two can communicate (in both directions)

Request Object

- `request.params` Parameters middleware
- `request.query` Extract query string parameter
- `request.route` Return currently-matched route
- `request.cookies` Cookies, requires cookie-parser
- `request.signedCookies` Signed cookies, requires cookie-parser
- `request.body` Payload, requires body-parser

http://expressjs.com/en/api.html#req

Request Object Examples

- request.query

Optional

```
http://localhost:3000/search?q=nodejs&lang=eng
{"q": "nodejs", "lang": "eng"}
```

- request.params

Mandatory

```
app.get('/api/:id/:name/:city',
        function(req, res) {
                res.end(req.params);
        }); // //
http://localhost:3000/api/1/Josh/Fairfield
{ id: 1, name: 'Josh', city: 'Fairfield' }
```

- request.body

```
app.use(bodyParser.urlencoded());
app.post('/api', function(req, res){
        res.end(req.body);
});
$ curl -i http://localhost:3000/api -d
'name=Josh&lastname=Edward'
{ name: 'Josh', lastname: 'Edward' }
```

Other Request Header Properties

`request.get(headerKey)`	Value for the header key
`request.accepts(type)`	Checks if the type is accepted
`request.acceptsLanguage(language)`	Checks language
`request.acceptsCharset(charset)`	Checks charset
`request.is(type)`	Checks the type
`request.ip`	IP address
`request.ips`	IP addresses (with trust-proxy on)
`request.path`	URL path
`request.host`	Host without port number
`request.fresh`	Checks freshness
`request.stale`	Checks staleness
`request.xhr`	True for AJAX-y requests
`request.protocol`	Returns HTTP protocol
`request.secure`	Checks if protocol is https
`request.subdomains`	Array of subdomains
`request.originalUrl`	Original URL

Response Object

- `response.redirect(status, url)` Redirect request
- `response.redirect(url)` Redirect to new path with status 302
- `response.send(status,data)` Send response
- `response.json(status, data)` Send JSON and force proper headers
- `response.jsonp(data)` JSON data will be wrapped in JS function call
- `response.sendfile(path, options, callback)` Send a file
- `response.render(templateName, locals, callback)` Render a template
- `response.locals` Pass data to template()
- `response.status(status)` Send status code

http://expressjs.com/en/api.html#res

Response Object Examples

```
// Passing Data to Templates
app.get('/api', function(req, res){
      res.locals = { title: 'CS572' };
      res.render('index');
});
app.get('/api', function(req, res, next){
      res.locals = { title: 'CS572' };
      return next();
}, function(req, res){
      res.render('index');
});
// another way to pass data to templates:
app.get('/render-title', function(req, res) {
      res.render('index', { title: 'CS572' })
});

// a common way to send status number
response.status(200).send('Welcome')
```

The `response.send()` method conveniently outputs any data application thrown at it (such as strings, JavaScript objects, and even Buffers) with automatically generated proper HTTP headers (Content-Length, ETag, or Cache-Control).

Resources

- EJS:
  - https://www.npmjs.com/package/ejs
  - https://ejs.co/
- Handlebars
  - https://handlebarsjs.com/
- Pug
  - https://pugjs.org/api/getting-started.html
- MVC: https://developer.mozilla.org/en-US/docs/Glossary/MVC

Homework

- Continue work on the same project and add features below:
  1. Add product to shopping cart
  2. Delete product from shopping cart
  3. Get shopping cart