Template Engines

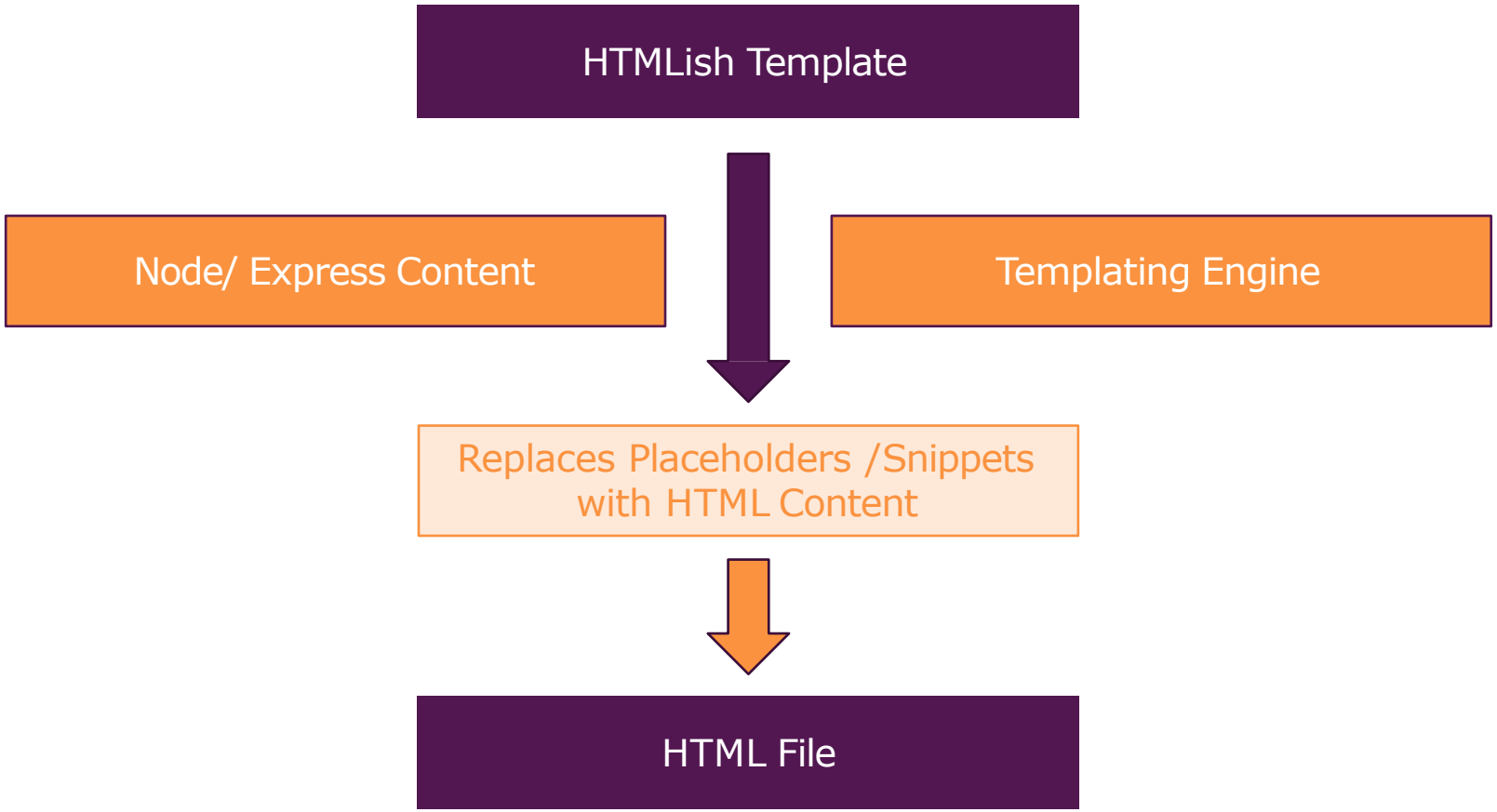
Template Engines

- Template engines are libraries that allow us to use different template languages (EJS, Pug..). (so that you can write separate HTML-like files for your view with special instructions to insert values into the HTML).

- 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

- **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
- **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
- **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

Main Point

- Templates allow you to separate your view logic into separate files. These files are basically HTML, but have additional special syntax to include variables and control statements.

- Science of Consciousness: Purification leads to Progress.

Overview of Pug Template Engine

Example HTML

```
<div>
  <h1>Ocean's Eleven</h1>
  <ul>
    <li>Comedy</li>
    <li>Thriller</li>
  </ul>
  <p>Danny Ocean and his eleven accomplices plan to rob
     three Las Vegas casinos simultaneously.</p>
</div>
```

Pug Version

```
div
  h1 Ocean's Eleven
  ul
    li Comedy
    li Thriller
  p.
    Danny Ocean and his eleven accomplices plan to rob
    three Las Vegas casinos simultaneously.
```

Simple Tags

- There are no "closing" tags in Pug.
- Instead, Pug uses indentation (i.e. white space) to determine how tags are nested.

```
div
  p Hello!
  p World!
```

- In the example above, since the paragraph tags are indented, they will end up inside the div tag.
- Pug compiles this accurately by treating the first word on each line as a tag, while subsequent words on that line are treated as text inside the tag.

Attributes

PUG:

```
div(class="movie-card", id="oceans-11")
  h1(class="movie-title") Ocean's 11
  img(src="/img/oceans-11.png", class="movie-poster")
  ul(class="genre-list")
    li Comedy
    li Thriller
```

HTML:

```
<div class="movie-card" id="oceans-11">
  <h1 class="movie-title">Ocean's 11</h1>
  <img src="/img/oceans-11.png" class="movie-poster">
  <ul class="genre-list">
    <li>Comedy</li>
    <li>Thriller</li>
  </ul>
</div>
```

Blocks of Text

- Adding a period (full stop) after your tag indicates that everything inside that tag is text and Pug stops treating the first word on each line as an HTML tag.

```
div
  p How are you?
  p.
    I'm fine thank you.
    And you? I heard you fell into a lake?
    That's rather unfortunate. I hate it when my shoes get wet.
```

JavaScript in Pug

- By starting a line with a hyphen, we indicate to the Pug compiler that we want to start using JavaScript.

PUG

```
- var x = 5;
div
  ul
    - for (var i=1; i<=x; i++) {
      li Hello
    - }
```

HTML

```
<div>
  <ul>
    <li>Hello</li>
    <li>Hello</li>
    <li>Hello</li>
    <li>Hello</li>
    <li>Hello</li>
  </ul>
</div>
```

JavaScript in Pug

- We use a hyphen when the code doesn't directly add output.
- If we want to use JavaScript to output something in Pug, we use =.

PUG

```
- var x = 5;
div
  ul
    - for (var i=1; i<=x; i++) {
      li= i + ". Hello"
    - }
```

HTML

```
<div>
  <ul>
    <li>1. Hello</li>
    <li>2. Hello</li>
    <li>3. Hello</li>
    <li>4. Hello</li>
    <li>5. Hello</li>
  </ul>
</div>
```

Looping in Pug

- Pug provides an excellent looping syntax so that you don't need to resort to JavaScript. Let's loop over an array:

PUG

```
- var droids = ["R2D2","C3PO","BB8"];
div
  h1 Famous Droids from Star Wars
  for name in droids
    div.card
      h2= name
```

HTML

```
<div>
  <h1>Famous Droids from
Star Wars</h1>
  <div class="card">
    <h2>R2D2</h2>
  </div>
  <div class="card">
    <h2>C3PO</h2>
  </div>
  <div class="card">
    <h2>BB8</h2>
  </div>
</div>
```

JavaScript Expressions

- Any JavaScript expression can be inserted by using `#{...}`.

PUG

```
- var profileName = "Danny Ocean";
div
  p Hi there, #{profileName}. How are you doing?
```

HTML

```
<div>
  <p>Hi there, Danny Ocean. How are you doing?</p>
</div>
```

Mixins

- Mixins are like functions. They take parameters as input and give markup as output:

```
mixin thumbnail(imageName, caption)
  div.thumbnail
    img(src="/img/#{imageName}.jpg")
    h4.image-caption= caption
```

- Once the mixin is defined, you can call the mixin with the + syntax:

```
+thumbnail("oceans-eleven", "Danny Ocean makes an elevator pitch.")
+thumbnail("pirates", "Introducing Captain Jack Sparrow!")
```

Mixins

- Which will output HTML like this:

```
<div class="thumbnail">
  <img src="/img/oceans-eleven.jpg">
  <h4 class="image-caption">
    Danny Ocean makes an elevator pitch.
  </h4>
</div>
<div class="thumbnail">
  <img src="/img/pirates.jpg">
  <h4 class="image-caption">
    Introducing Captain Jack Sparrow!
  </h4>
</div>.
```

Using Pug

- Install pug into your project using NPM as below:

```
npm install pug
```

- Pug template must be written inside **`.pug`** file.

- All **`.pug`** files must be put inside **`/views`** folder in the root folder of Node.js application.

Using Pug with Express.js

- Assume the pug template is stored in file **`/views/sample.pug`**.

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

//set view engine
app.set("view engine","jade")

app.get('/', function (req, res) {
    res.render('sample');  //sends HTML version of sample.pug to Browser
});

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

Passing Data to Templates

```
// Passing Data to Templates
app.get('/api', function(req, res){
      res.locals = { title: 'CS572' };
      res.render('index');
});

// another way to pass data to templates
app.get('/render-title', function(req, res) {
      res.render('index', { title: 'CS572' })
});
```

Main Point

- The response object has a render() method that can be used to specify a template file. The local variables for the template can either be passed in as a second parameter to render() or get set beforehand on response.locals.

- *Science of Consciousness*: Every action has an equal and opposite reaction.

File System Module

Read File Example

- Assume we have the following HTML file:

demofile1.html

```
<html>
  <body>
    <h1>My Header</h1>
    <p>My paragraph.</p>
  </body>
</html>
```

Read File Example

- Create a Node.js file that reads the HTML file, and returns the content:

demo_readfile.js

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Read File Example (con't)

- Run the node.js file:

```
C:\Users\Your Name>node demo_readfile.js
```

- Use the following URL in your Browser: http://localhost:8080

- The result should be as follows:

My Header

My paragraph.

Append File Example

- The **`fs.appendFile()`** method appends specified content to a file. If the file does not exist, the file will be created:

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

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err)
{
  if (err) throw err;
  console.log('Saved!');
});
```

Write File Example

- The **`fs.writeFile()`** method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

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

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err)
{
  if (err) throw err;
  console.log('Saved!');
});
```

Delete File Example

- To delete a file with the File System module, use the **`fs.unlink()`** method:

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

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Read File Synchronously

- Use **`fs.readFileSync()`** method to read file synchronously as shown below:

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

var data = fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
```

- Notice that no callback function is required for a synchronous read.
- The **`console.log`** instruction is not executed until the read operation is complete.

Database Access

Install Database Driver

- Node.js supports all kinds of databases, no matter if it is a relational database or NoSQL database.
- To access the database from Node.js, you first need to install drivers for the database you want to use.
- For MS SQL Server use the following:

```
npm install mssql
```

MS SQL Server Example

- After installing the driver, we are ready to access MS SQL server database.
- We will connect to a local SQLExpress database server and fetch all the records from *Student* table in *SchoolDB* database.
- The Student Table has fields: *StudentID* and *StudentName*.

Access Database using Express

- Define the Database Configuration:

server.js

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

app.get('/', function (req, res) {

    var sql = require("mssql");

    // config for your database
    var config = {
        user: 'sa',
        password: 'mypassword',
        server: 'localhost',
        database: 'SchoolDB',
        options: { trustServerCertificate: true }
    };
```

Access Database using Express

- Connect to the Database:

server.js (continued)

```
// connect to your database
    sql.connect(config, function (err) {

        if (err) console.log(err);

        // create Request object
        var request = new sql.Request();
```

Query the Database

server.js (continued)

```
 // query to the database and get the records
        request.query('select * from Student', function (err, recordset) {

            if (err) console.log(err)

            // send records as a response
            res.send(recordset);

        });
    });
});

var server = app.listen(5000, function () {  // start the server
    console.log('Server is running..');
});
```

Access MS SQL Databae

- Run the above example using **`node server.js`** command and point your browser to http://localhost:5000.

- This will display an array of all students from Student table.

Use Pug Template to Display Student List

- Insert the following into the server.js program: `app.set("view engine","pug")`
- Put following Pug Template in "views" folder:

StudentList.pug

```
doctype html
html
head
title=title
body
    h1 Student List using Jade engine

    ul
        each item in studentList
            li=item.StudentName
```

Use Pug Template to Display Student List

- Replace the final **`res.send(recordset)`** with the following:

```
res.render('StudentList', { studentList: recordset.recordsets[0] });
```

Resources

- Pug
  - https://pugjs.org/api/getting-started.html
  - https://www.sitepoint.com/jade-tutorial-for-beginners/
- MS SQL Server Database Access
  - https://www.tutorialsteacher.com/nodejs/access-sql-server-in-nodejs