



Package Management(Node.js)

Deloitte Technology Academy (DTA)



Agenda

Topics	Descriptions	Duration
Node and Node Package Manager (NPM) Basics	<ul style="list-style-type: none">• Introduction, Package.json, Framework, Tools, and NPM Commands• Working With Modules, Require Function, Built-In Global Variables, and Running Scripts	6 hours
Webpack	Node Environment, Project Bundling, Serving, and Building Application Using Webpack	

</>

Learning Objectives

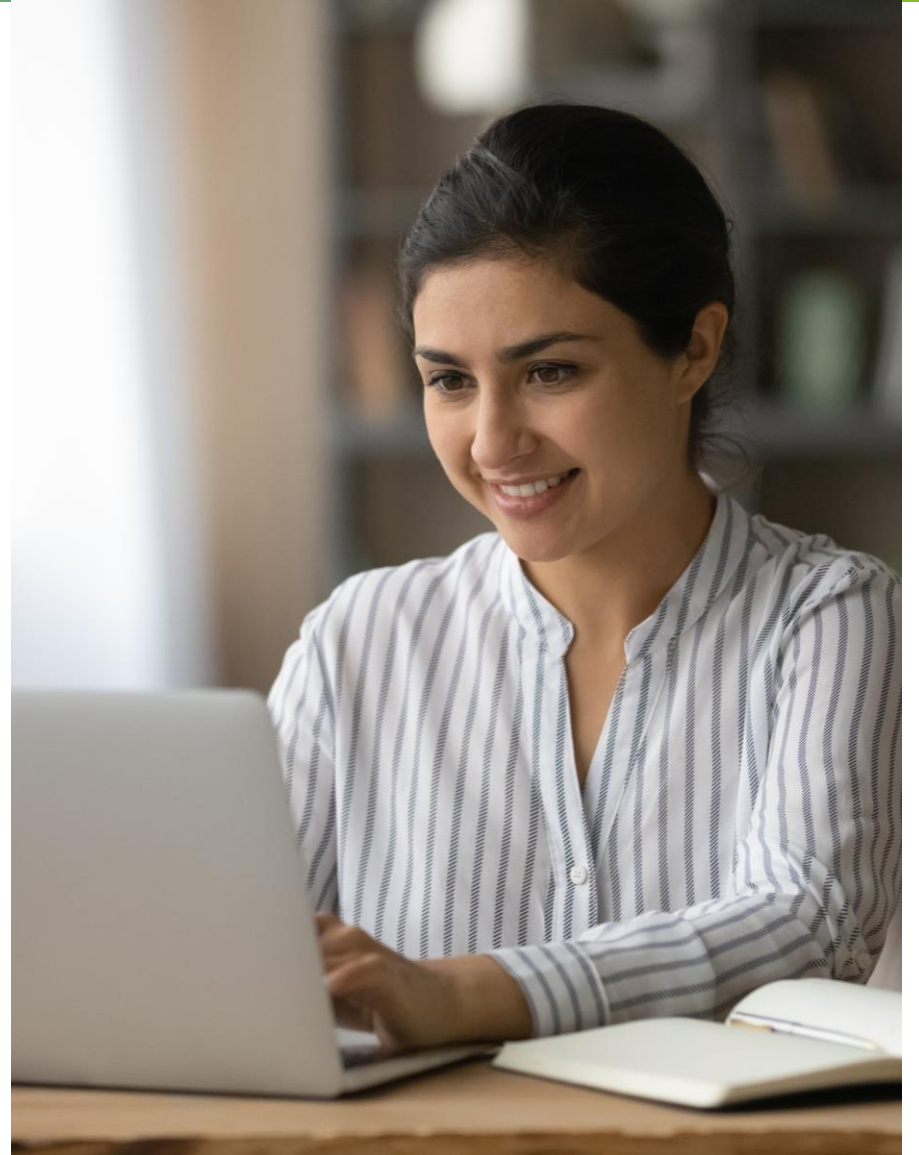
By the end of this session, you will be able to:

- Explain Node Package Manager (NPM) along with the frameworks and tools it uses
- Explain webpack and learn how to bundle and build applications using webpack



</>

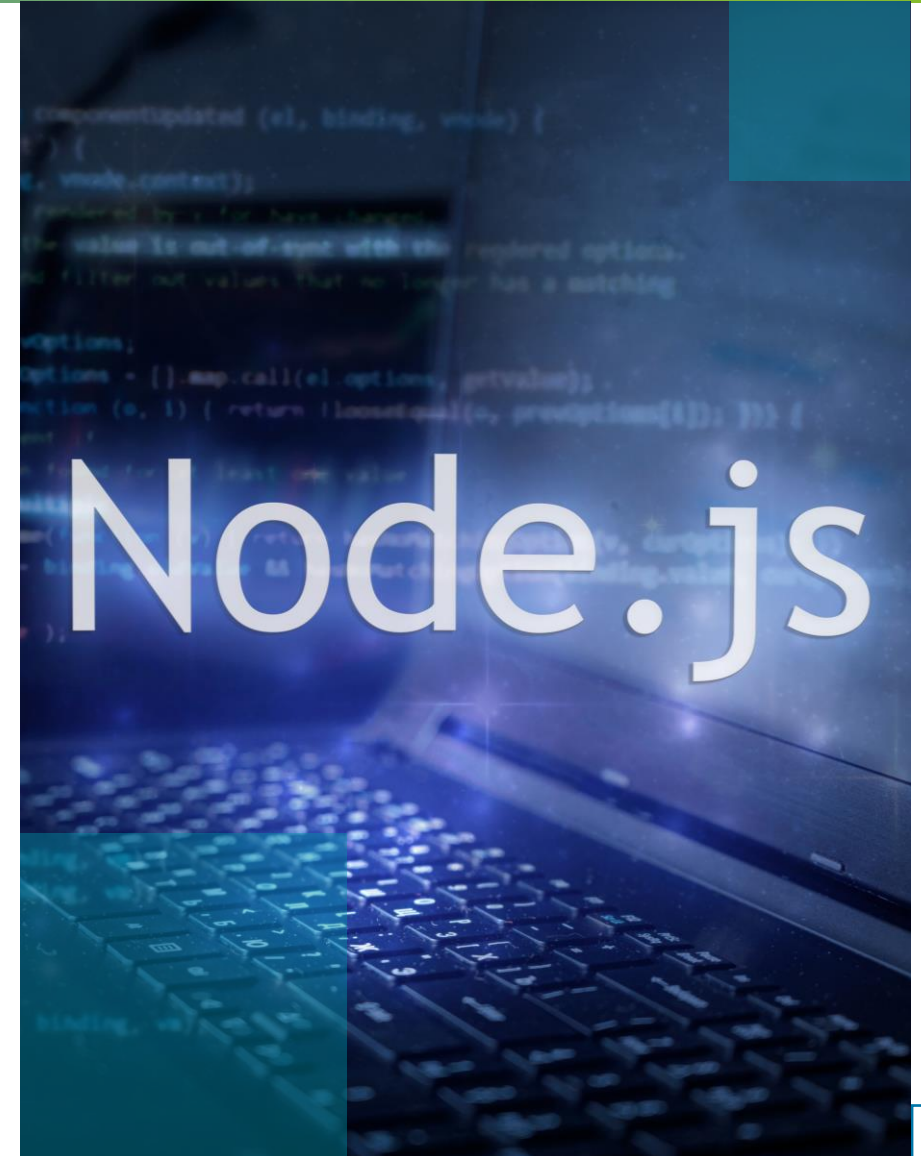
Node and NPM Basics



</>

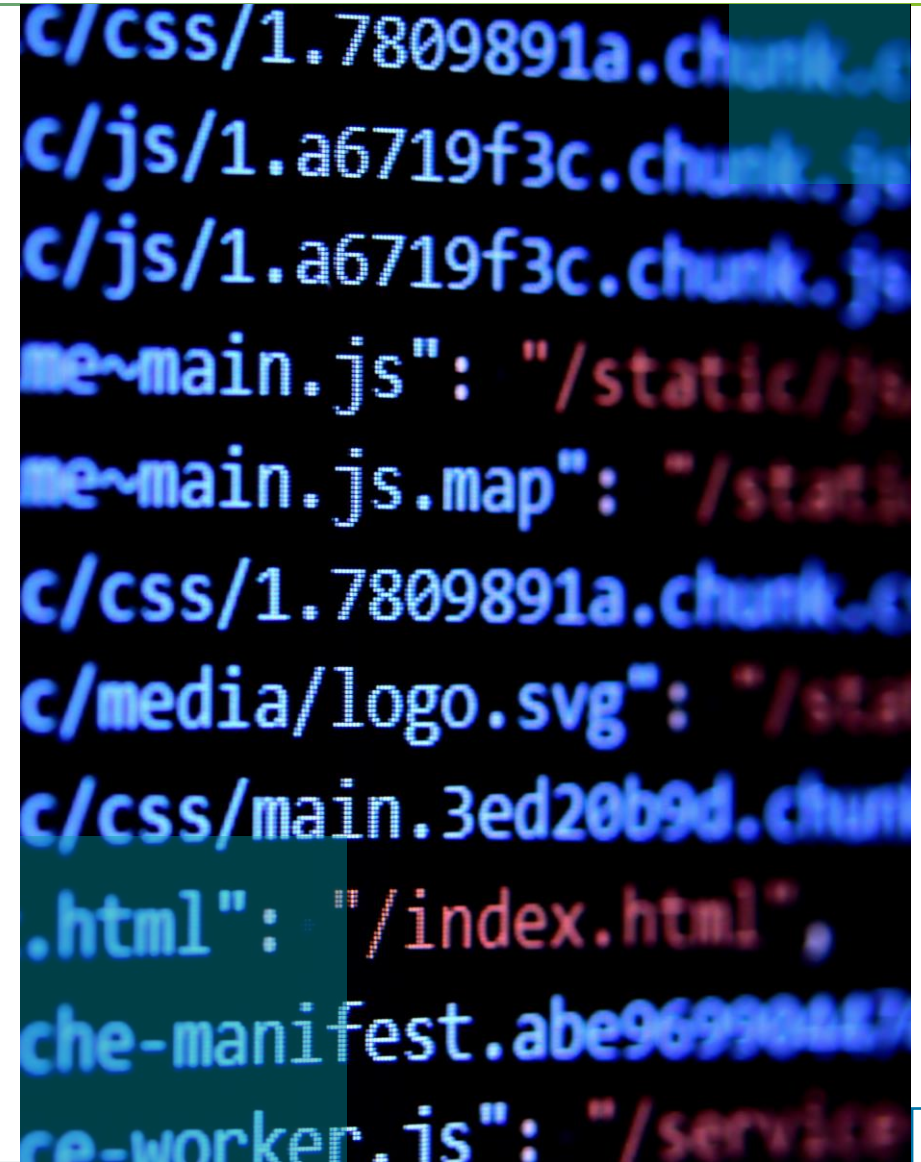
Introduction

- Node.js is an open-source server environment.
- Node.js uses JavaScript on the server.
- NPM is the standard package manager for Node.js.
- It started as a way to download and manage dependencies of Node.js packages, but it has since become a tool also used in frontend JavaScript.
- NPM can manage packages that are local dependencies of a particular project, as well as globally-installed JavaScript tools.
- In addition to plain downloads, NPM also manages versioning, so you can install any version, higher or lower according to the needs of your project.



Package.json

- The package.json file is the heart of any Node project.
- It is a JavaScript Object Notation (JSON) file that lives in the root directory of the project.
- The package.json holds important information about the project.
- It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number, and a list of dependencies required by the application.

A blurred image showing a list of files and their relative paths, likely from a package.json or a similar manifest file. The text is color-coded, with blue for file names and red for paths. Visible entries include: c/css/1.7809891a.chunk.js, c/js/1.a6719f3c.chunk.js, me~main.js, me~main.js.map, c/css/1.7809891a.chunk.js, c/media/logo.svg, c/css/main.3ed20b9d.chunk.js, .html, che-manifest.abe96990483, and ce-worker.js.

```
c/css/1.7809891a.chunk.js
c/js/1.a6719f3c.chunk.js
c/js/1.a6719f3c.chunk.js
me~main.js": "/static/js
me~main.js.map": "/static
c/css/1.7809891a.chunk.js
c/media/logo.svg": "/sta
c/css/main.3ed20b9d.chunk
.html": "/index.html",
che-manifest.abe96990483
ce-worker.js": "/service
```

Package.json (Cont.)

Sample Package.json is shown below:

```
{
  "name": "del_package",
  "description": "",
  "version": "1.0.0",
  "scripts": {
    "start": "node index.js"
  },
  "repository": {
    "type": "test",
    "url":
```

```
"https://test.com/test/del_package.git"
  },
  "keywords": [],
  "author": "Deloitte",
  "bugs": {
    "url": "https://test/issues"
  },
  "homepage": "https://test/del_package"
}
```



NPM commands

Below are some of the most common NPM Commands:

Commands	Description
<code>npm install</code>	Install package.json dependencies
<code>npm list -g</code>	List globally installed packages
<code>npm -g uninstall</code>	Uninstall global package
<code>npm-windows-upgrade</code>	Upgrade npm on Windows
<code>npm run</code>	list available scripts to run

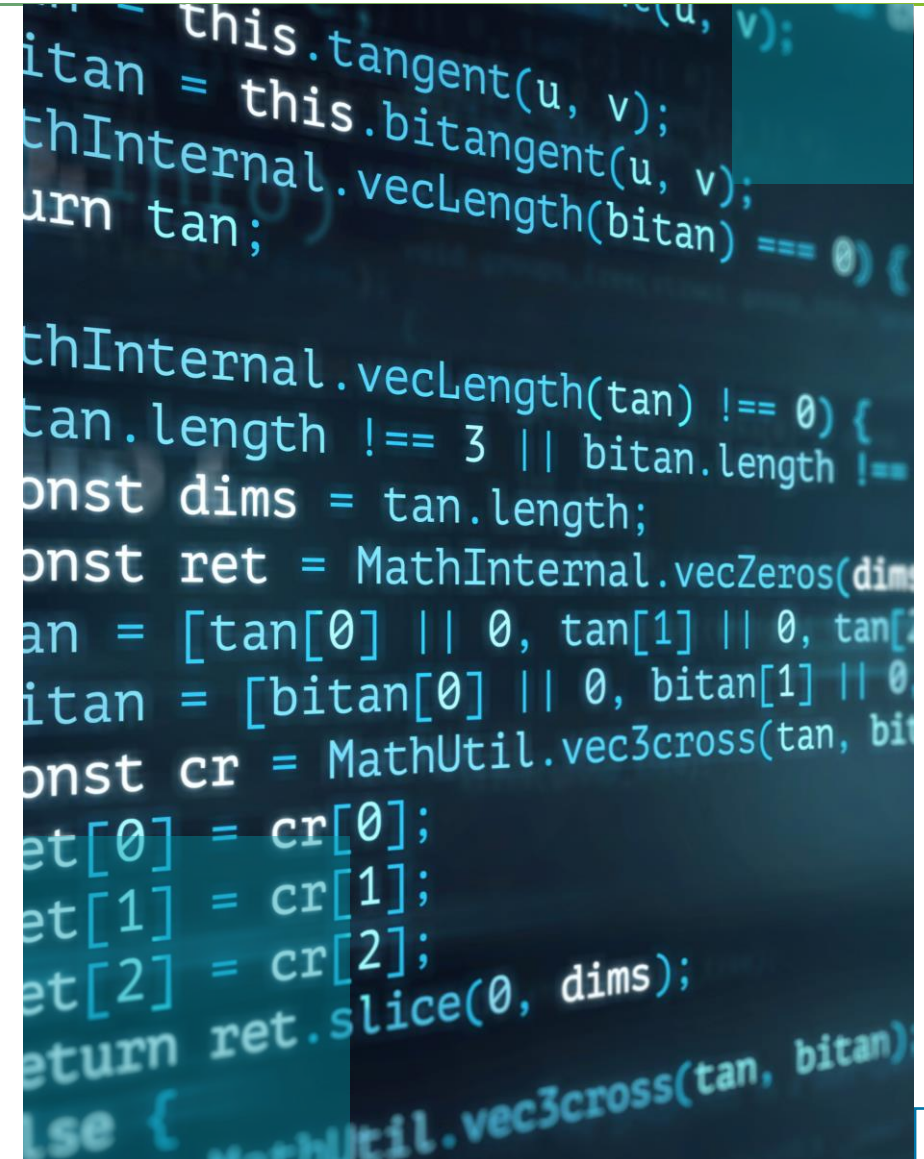
Working With Modules

- Modules are same as JavaScript libraries. They are a set of functions, to include in an application.
- Node.js has a set of built-in modules which you can use without any further installation.
- To include a module, use the `require()` function with the name of the module:
`var http = require('http');`
- We can create our own modules, and easily include them in our applications.
- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- We can use the `createServer()` method to create an HTTP server.

```
var http = require('http');  
//create a server object:  
http.createServer(function (req, res) {  
  res.write('Hello World!'); //response to the client  
  res.end();  
}).listen(8080); // server object listens on port 8080
```

Require Function

- `require()` is used to consume modules. It allows you to include modules in your app. You can add built-in core Node.js modules, community-based modules (`node_modules`), and local modules too.
- The main object exported by `require()` module is a function. When Node invokes that `require()` function with a file path as the function's only argument.
- Node.js follows the CommonJS module system, and the built-in `require` function is the easiest way to include modules that exist in separate files. The basic functionality of `require` is that it reads a JavaScript file, executes the file, and then proceeds to return the exports object.
- Example: `const mod = require('./module1.js')`



```
this.tangent(u, v);  
itan = this.bitangent(u, v);  
thInternal.vecLength(bitan) === 0) {  
    return tan;  
  
    thInternal.vecLength(tan) !== 0) {  
        tan.length !== 3 || bitan.length !==  
        const dims = tan.length;  
        const ret = MathInternal.vecZeros(dims);  
        an = [tan[0] || 0, tan[1] || 0, tan[2] || 0];  
        itan = [bitan[0] || 0, bitan[1] || 0, bitan[2] || 0];  
        const cr = MathUtil.vec3cross(tan, bitan);  
        ret[0] = cr[0];  
        ret[1] = cr[1];  
        ret[2] = cr[2];  
        return ret.slice(0, dims);  
    }  
    use { MathUtil.vec3cross(tan, bitan)
```

Built-In Global Variables

- Node.js has many built-in global identifiers.
- Some of these global identifiers are true globals, being visible everywhere; while others exist at module level, and are inherent to every module, thus being pseudo-globals.

Below is the list of 'true globals':

Global

The global namespace. Setting a property to this namespace makes it globally visible within the running process.

Process

The Node.js built-in process module, which provides interaction with the current Node.js process.

Console

The Node.js built-in console module, which wraps various STDIO functionality in a browser-like way. `setTimeout()`, `clearTimeout()`, `setInterval()`, `clearInterval()`
The built-in timer functions are globals.

</>

Built-In Global Variables (Cont.)

There are several 'pseudo-globals' included at the module level in every module as given below:

module, module.exports, exports	All these objects pertain to the Node.js module system.
__filename	The __filename keyword have the path of the currently executing file. This is not defined when we run the Node.js Read-Eval-Print-Loop (REPL).
__dirname	Like __filename, the __dirname keyword will have the path of the root directory of the script currently executing.
require()	The require() function is a built-in function which is exposed per module, that allows other valid modules to be included.

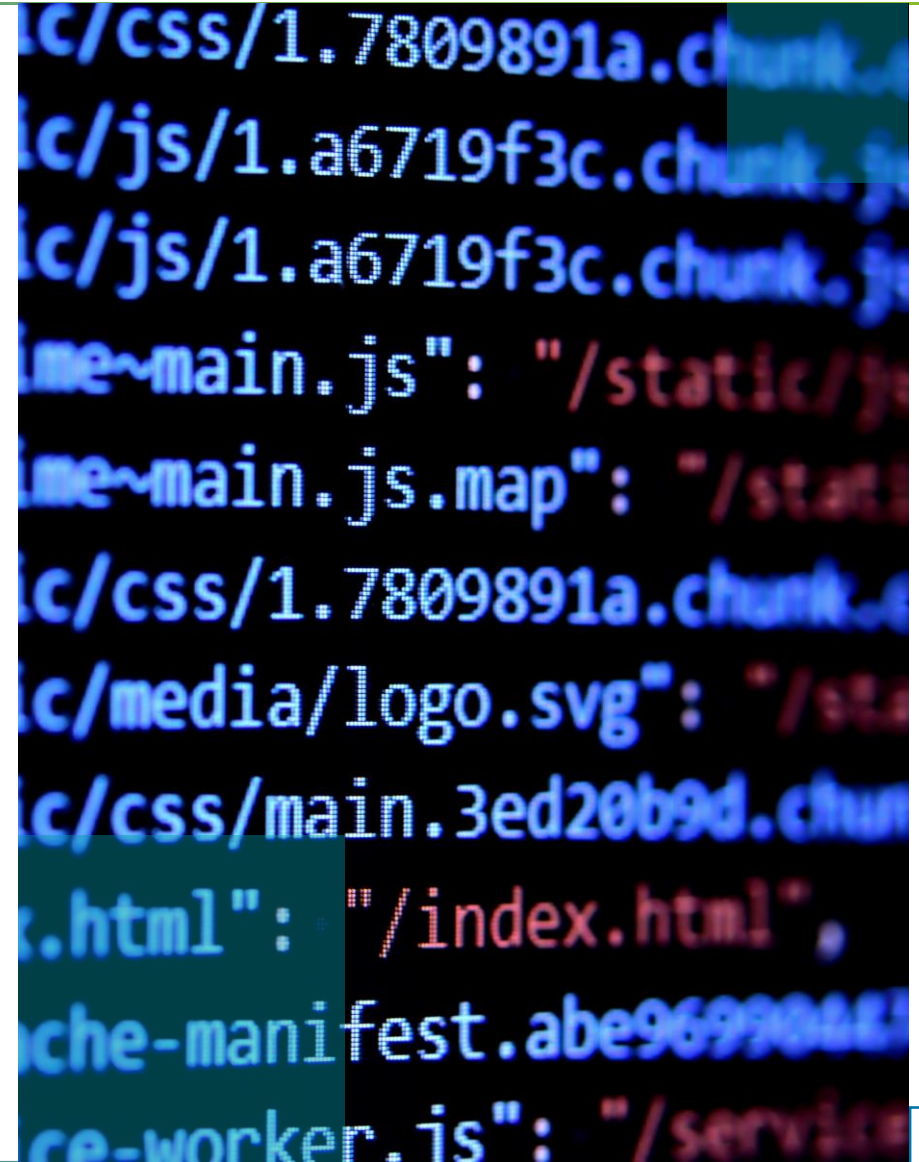
</>

Webpack



Node Environment

- Webpack is a module bundler.
- The main purpose of webpack is to bundle JavaScript files for usage in a browser, yet it is also capable of transforming, bundling, or packaging just about any resource or asset.
- The webpack command line environment option `--env` allows you to pass in as many environment variables as you like.
- Environment variables will be made accessible in your `webpack.config.js`. For example, `--env production` or `--env goal=local`.
- Webpack Command-Line Interface (CLI) offers some built-in environment variables which you can access inside a webpack configuration.



Project Bundling

- To bundle a Project, we'll tweak our directory structure slightly first, separating the "distribution" code (./dist). from our "source" code (./src).
- The "source" code is the code which we'll edit and write.
- The "distribution" code is the optimized and minimized output of our build process that will eventually be loaded in the browser.

Tweak the directory structure as shown below:

```
webpack-demo
|- package.json
|- package-lock.json
|- /dist
|  |- index.html
|- index.html
|- /src
   |- index.js
```


</>

Summary

Here are the key learning points of the module.

- NPM is the standard package manager for the Node JavaScript platform.
- Webpack's main purpose is to bundle JavaScript files for usage in a browser.



Thank You



About Deloitte

Deloitte refers to one or more of Deloitte Touche Tohmatsu Limited, a UK private company limited by guarantee (“DTTL”), its network of member firms, and their related entities. DTTL and each of its member firms are legally separate and independent entities. DTTL (also referred to as “Deloitte Global”) does not provide services to clients. In the United States, Deloitte refers to one or more of the US member firms of DTTL, their related entities that operate using the “Deloitte” name in the United States and their respective affiliates. Certain services may not be available to attest clients under the rules and regulations of public accounting. Please see www.deloitte.com/about to learn more about our global network of member firms.

This communication contains general information only, and none of Deloitte Touche Tohmatsu Limited (“DTTL”), its global network of member firms or their related entities (collectively, the “Deloitte organization”) is, by means of this communication, rendering professional advice or services. Before making any decision or taking any action that may affect your finances or your business, you should consult a qualified professional adviser.

No representations, warranties or undertakings (express or implied) are given as to the accuracy or completeness of the information in this communication, and none of DTTL, its member firms, related entities, employees or agents shall be liable or responsible for any loss or damage whatsoever arising directly or indirectly in connection with any person relying on this communication. DTTL and each of its member firms, and their related entities, are legally separate and independent entities.