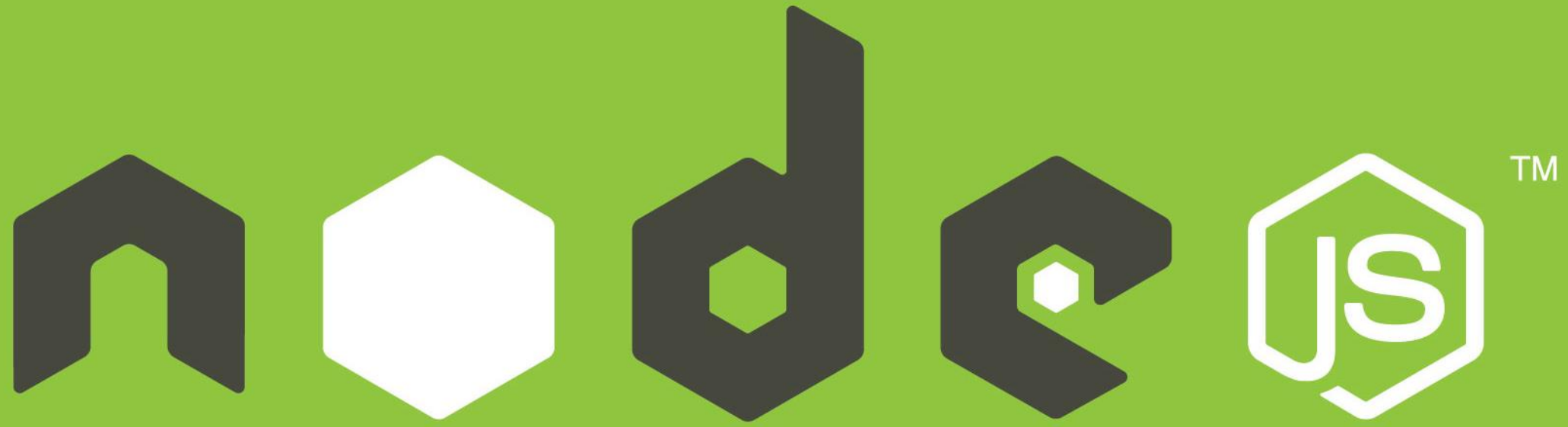
NodeJS, NPM, Yarn etc.

10. april 2018

Agenda

- Node.js
- NPM
- NuGet
- Gulp
- Webpack



Node.js – Server side JavaScript

What is Node.js

An open-source, cross-platform JavaScript runtime environment

Intended for developing serverside tools and applications

Event driven, works on the Non-Blocking Modell

Used by: GoDaddy, IBM, Microsoft, Paypal

Uses Google V8 JavaScript engine, to compile to native machine code

I.E. it does not interpret it real time (faster)

JavaScript on the Server side

Basically NodeJS allows us to run JavaScript on the backend

Using Google V8

Has a runtime environment and use NPM for dependencies

Where to use NodeJS

Few CPU cycles

I/O operations

Chat/Messaging services

Real-time applications

Communication hubs

High concurrency applications

Where not to use NodeJS

Heavy computation

Large and complicated web applications

That’s it!

No more information on Node.js

Had to be mentioned because NPM runs on Node.js

Please go to https://nodejs.org and download NodeJS LTS version

Node Package Manager

What is NPM

Default package manager for Node.js

It allows users to consume and distribute JavaScript modules

Packages on NPM are in CommonJS format and include a metadata file in JSON format

Over 280,000 packages are available

Created by Isaac Z. Schlueter

Free and open-source!

Usage

NPM can handle project specific dependencies

Likewise it can handle globally installed modules

Project specific modules are saved in node_modules folder

Is normally used from commandline (Visual Studio handles it for us)

Modules for a project are managed through Package.json

Understanding *package.json*

It is a valid JSON object

Name and *Version* fields are required

- The combination makes a unique identifier for a given package

Other fields

- Description
- Keywords
- Homepage
- Bugs
- License
- Author and contributors
- Dependencies

Example: Package.json

```
{
  "name": "ASSIGNMENTONE", // Project name
  "version": "0.0.1", // Version number
  "description": "Webpack + ASP.NET MVC 4 sample for ReactJS.NET",
  "author": "Nicolai Oksen", // Who made it
  "license": "BSD", // Which opensource license applies
  "devDependencies": { // Development dependencies
    "gulp": "^3.9.1",
    "gulp-concat": "^2.6.1",
    "gulp-uglify": "^2.1.0",
  },
  "dependencies": { // Dependencies for project
    "react": "^15.4.2",
    ...
  }
}
```

Where to find packages?

https://npmjs.org

Installing packages

Npm is installed together with NodeJS

To install packages, you have to use a Command Line Interface (CLI)

Packages can be installed on a per project basis or globally

Global packages are available globally

Example: Command line

```
// Install module into node_modules folder. No entry in Package.json
npm install react

// Install module and save dependency in Package.json dependencies
npm install --save react

// Install mdole and save dependency in package.json under devDependencies
npm install --save-dev react

// Install module globally
npm install --global react
```

NPM alternative

Yarn! ☺

https://yarnpkg.com/lang/en/docs/install/

Example: Visual Studio Code

Exercise time

→ Open code example for this Lesson (7)

→ Open the whole folder in VS Code

→ Open package.json file

→ Go through the file

→ Install all packages via command line

Honorable mention: NuGet

What is NuGet

Free and open source package manager

Intended for the Microsoft Development platform

Comes pre-installed on Visual Studio 2012 and above

Supports the .NET Framework and C++

Usage

Demo time...

Gulp
Javascript Task Runner

What is Gulp

An open-source JavaScript toolkit

A streaming build system in front-end web development.

It is a task runner

It is built on Node.js and Node Package Manager (npm)

Used for automation of time-consuming and repetitive tasks

Focuses on ”Code-over-configuration”

Has over 300 plugins

Usage

Gulp is run from the command line

When using Visual Studio it is possible to use the Task Runner Explorer tab

It requires a package.json and a gulpfile.js in the root directory

All necessary gulp plugins should be installed into ”devDependencies” in package.json

The gulpfile.js - Plugins

All required plugins are *required* at the top:

```
var gulp = require('gulp');
var util = require('gulp-util');
```

Or like this

```
var gulp = require('gulp'),
    util = require('gulp-util');
```

The gulpfile.js - Tasks

Tasks can then be created.

First parameter is the name and second is a function

```
gulp.task('taskName1', function () {
    //do something
});
```

Tasks can aslo perform several predefined function calls

```
function fn1() {…}
function fn2() {…}
// Task with array of function names
gulp.task('taskName2', ['fn1', 'fn2']);
```

The gulpfile.js – Default task

Should be at the end of the gulpfile.js

Can be run by typing ”`gulp`” in the command line

```
gulp.task('default', [‘taskName1', ‘taskName2']);
```

Example: Gulp task

Task that handles the bundling of react for my project

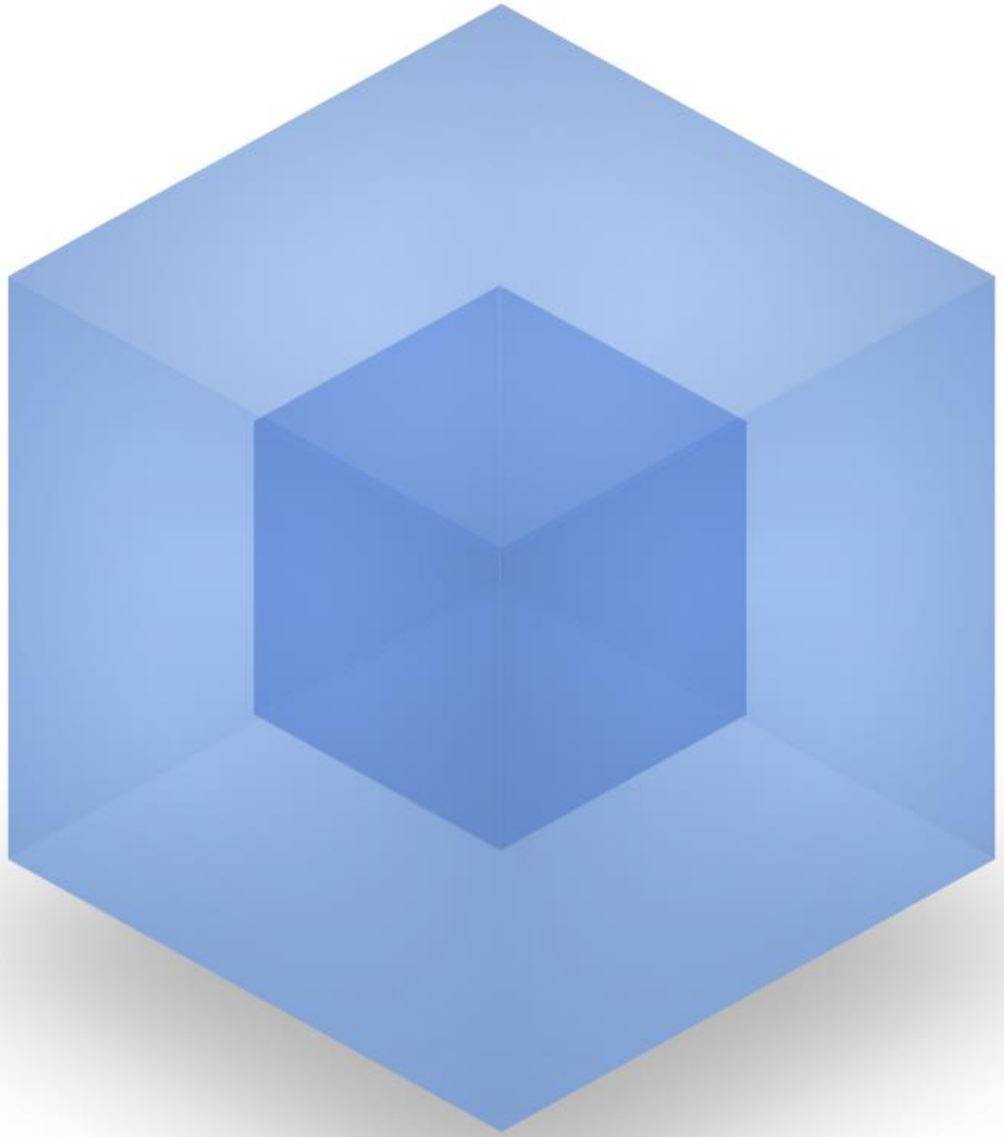
```
gulp.task('scripts', function () {
    return browserify()
        .external(dependencies) // Load dependencies from node_modules
        .bundle() // Bundles them together
        .on('error', handleErrors) // Uses a function if something fails
        .pipe(source('main.js')) // Creates a new file called main.js
        .pipe(gulp.dest('./wwwroot/build/')); // Outputs it to directory
});
```

Example: Gulp Watch Task

Monitors directories and files for changes. Acts accordingly.

```
gulp.task('watch', function () {
    // libs task runs when changes occurs
    gulp.watch('package.json', ['libs']);
    // Monitors the src folder
    // scripts task runs when changes occurs
    gulp.watch('src/**', ['scripts']);
});
```

Webpack
Module bundler



What is webpack

It is a module bundler

Intended to allow big projects to function better

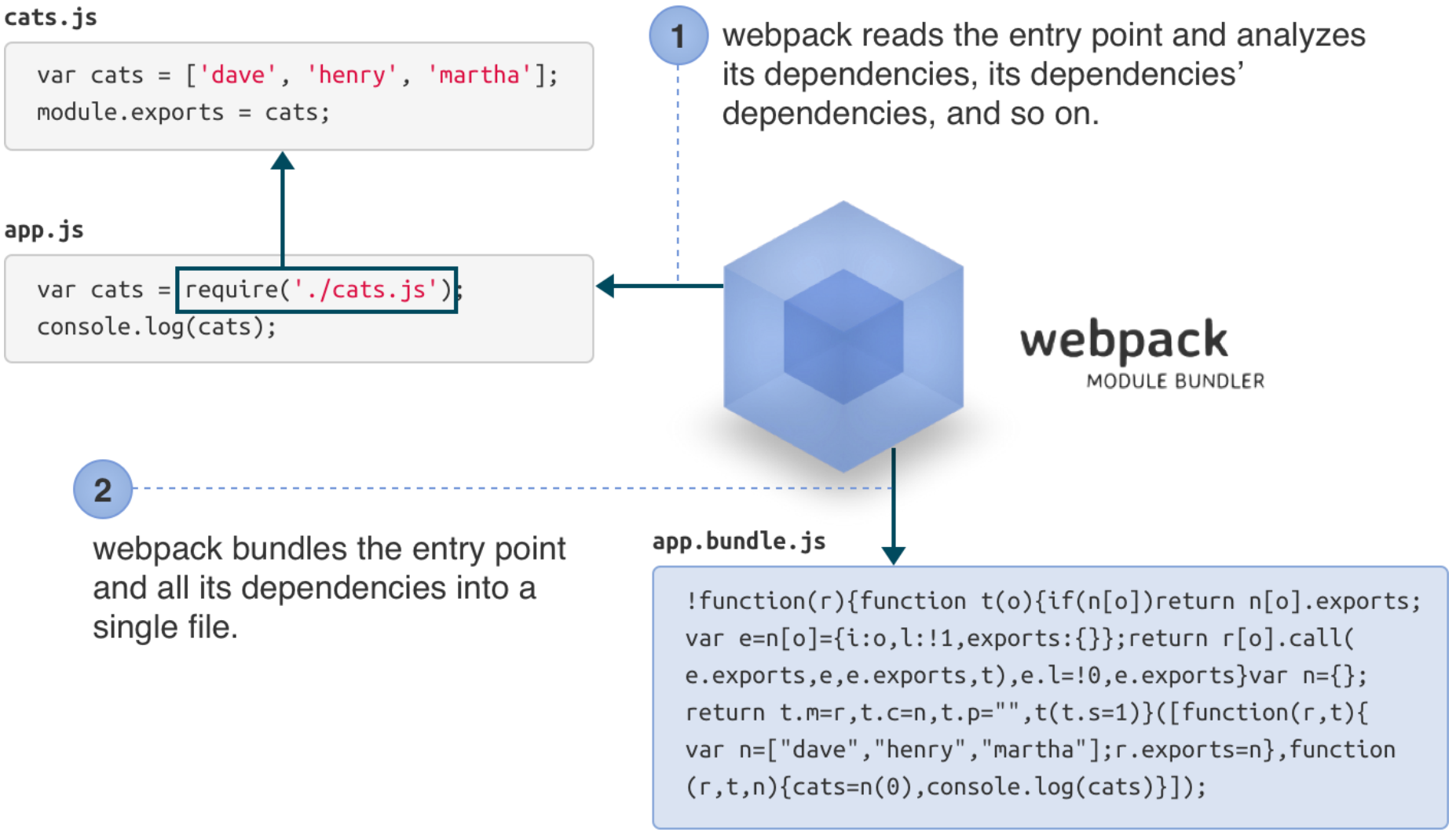
Ability to customize nearly every part of the module bundler

Ability to integrate 3rd-party libraries as modules

Keep initial loading time low

Every static asset should be able to be a module

Usage


cats.js
var cats = ['dave', 'henry', 'martha'];
module.exports = cats;
app.js
var cats = require('./cats.js');
console.log(cats);
1
webpack reads the entry point and analyzes its dependencies, its dependencies' dependencies, and so on.
webpack
MODULE BUNDLER
2
webpack bundles the entry point and all its dependencies into a single file.
app.bundle.js
!function(r){function t(o){if(n[o])return n[o].exports;
var e=n[o]={i:o,l:!1,exports:{}};return r[o].call(
e.exports,e,e.exports,t),e.l=!0,e.exports}var n={};
return t.m=r,t.c=n,t.p="",t(t.s=1)}([function(r,t){
var n=["dave","henry","martha"];r.exports=n},function
(r,t,n){cats=n(0),console.log(cats)}]);

Usage (cont.)

```
// cats.js <-- File
var cats = ['dave', 'henry', 'martha'];
module.exports = cats;

// App.js <-- File
cats = require('./cats.js');
console.log(cats);

// Command line
webpack./app.js app.bundle.js
```

Webpack.config.js

The most basic webpack configuration file:

```
module.exports = {
    entry: './src/app.js',
    output: {
        path: './js',
        filename: 'app.bundle.js'
    }
};
```

Webpack loaders

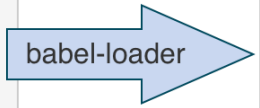
Initially webpack only supports JavaScript

Loaders allows support for multiple other things such as TypeScript, JSON etc.

This allows us to use ECMAScript 2015!

./App.js

```
import React from 'react'
import { Provider } from 'react-redux'
import AppRouter from './AppRouter'
export const App = ({ store }) => (
  <Provider store={store}>
    <AppRouter />
  </Provider>
)
```



babel-loader!./App.js

```
'use strict';
Object.defineProperty(exports, "__esModule", {
  value: true
});
exports.App = undefined;
var _react = require('react');
var _react2 = _interopRequireDefault(_react);
var _reactRedux = require('react-redux');
var _AppRouter = require('./AppRouter');
var _AppRouter2 = _interopRequireDefault(_AppRouter);
function _interopRequireDefault(obj) { return obj && obj.__esModule
  ? obj : { default: obj }; }
var App = exports.App = function App(_ref) {
  var store = _ref.store;
  return _react2.default.createElement(
    _reactRedux.Provider,
    { store: store },
    _react2.default.createElement(_AppRouter2.default, null)
  );
};
```

Webpack loaders (cont.)

```
module: {
    loaders: [{
        test: /\.js$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
    }]
}
```

We exclude node_modules!

Using babel-loader allows:

Specific imports

```
import React, {Component, PropTypes} from 'react'
import Login from './Login'
import Copyright from './Copyright'
```

And…

```
class Login extends Component {
    render () {
        ...
        )
    }
}
```

ES6 Features

Arrow functions

Classes

Template Strings

Destructuring

Collection spreading

Let + Const

…

Read more: https://github.com/lukehoban/es6features

Code Splitting

Split the dependency tree into smaller chunks and load on demand

- As opposed to loading everything up front

Code splitting helps in reducing

- Load time
- HTTP Requests
  - When using HTTP/2, chunk size is also a factor
- Optimizing the files to avoid code redundancies

Profiling

WebPack also allows for profiling

Basically analysing code and runtime for problem etc.

Multiple entrypoints

Multiple entry point allows webpack to create multiple bundles at once.

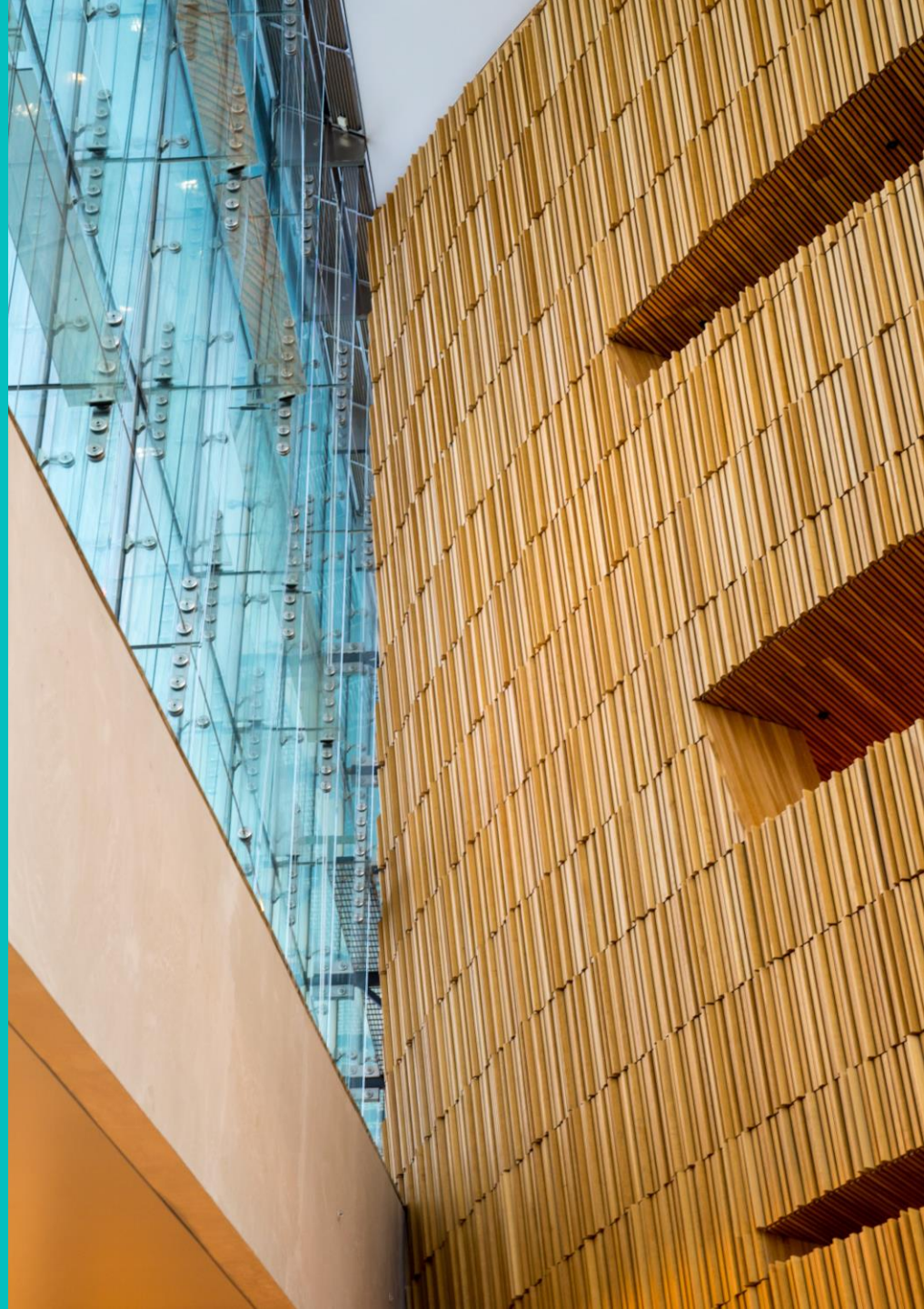
Hot Module Replacement

HMR is a way of exchanging modules in a running application

- Both adding and removing
- Allows updating changed modules without a full page reload

HMR has to enabled explicitly

Getting setup fast



Now you have some insight into how the different systems work.

Now for how to getting up and running in a jiffy! ☺

VueJS

Go to a command line

```
npm install -g @vue/cli

// or

yarn global add @vue/cli
```

Afterwards run

```
vue create my-project
```

Vue CLI Demo time

ReactJS

Go to a command line

```
npx create-react-app my-app
```

If you have a space in your user name on windows, use this:

```
npm install -g create-react-app

create-react-app my-app
```

Create React App Demo time

Dotnet

Go to a command line

```
dotnet new react
```

Then run

```
yarn install

// or

npm install
```

DotNet CLI Demo time

Questions?