MEAN Stack

COMP6205: Web Development

Dr A. Rezazadeh
November 17
Components of MEAN stack

- **MongoDB**
  - Document-based (JSON) NoSQL database powered by JavaScript
- **ExpressJS**
  - JavaScript back-end framework
- **AngularJS**
  - JavaScript front-end framework
- **NodeJS**
  - JavaScript run-time (C/C++ taps into Chrome V8 engine)
# MEAN vs. Microsoft

<table>
<thead>
<tr>
<th>Tool</th>
<th>MEAN stack</th>
<th>Microsoft Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>JavaScript (CSS, HTML, JSON)</td>
<td>C#, JavaScript (CSS, HTML, JSON)</td>
</tr>
<tr>
<td>IDE</td>
<td>WebStorm IDE (Sublime, Atom)</td>
<td>Visual Studio</td>
</tr>
<tr>
<td>Database</td>
<td>MongoDB</td>
<td>SQL Server</td>
</tr>
<tr>
<td>Framework</td>
<td>Express</td>
<td>.Net</td>
</tr>
<tr>
<td>Front-end</td>
<td>Angular</td>
<td>HTML, CSS</td>
</tr>
<tr>
<td>Web server</td>
<td>Node</td>
<td>IIS – Internet Information Server</td>
</tr>
<tr>
<td>Source control</td>
<td>GitHub</td>
<td>TFS – Team Foundation Server</td>
</tr>
</tbody>
</table>
Common Language & Data Format Throughout the MEAN Stack

- **Database**: MongoDB
  - Data format: BSON
  - Exposed data format: JSON
- **Application server**: Node.js and Express
  - Language: JavaScript
  - Data format: JSON
- **Front end**: AngularJS
  - Language: JavaScript
  - Data format: JSON
MongoDB

- Developed by MongoDB, Inc (initially called 10gen)
- It is a NoSQL database
- A document-oriented database
- It uses BSON format (Binary form of JSON)
  - JSON: JavaScript Object Notation.
  - JSON is a syntax for storing and exchanging data.
  - JSON is an easier-to-use alternative to XML.
The following JSON example defines an employees object, with an array of 3 employee records:

```json
{"employees": [  {
    "firstName": "John",  "lastName": "Doe"},
  {
    "firstName": "Anna",  "lastName": "Smith"},
  {
    "firstName": "Peter",  "lastName": "Jones"}
]}
```

More details: http://www.json.org/
XML Example

- The following XML example also defines an employees object with 3 employee records:

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```
What is MongoDB?

- Scalable High-Performance Open-source, Document-orientated database.
- Built for Speed
- Full Index Support for High Performance.
- Replication and Failover for High Availability.
- Auto Sharding for Easy Scalability.
- Map-Reduce for Aggregation.
  - Map-reduce is a data processing paradigm for condensing large volumes of data into useful *aggregated* results.
Sharding

Key Range
0..30

Key Range
31..60

Key Range
61..90

Key Range
91..100

Primary

Secondary

Secondary

Secondary

Primary

Secondary

Secondary

Secondary

Config

Config

Config
What is MongoDB?

- SQL was invented in the 70’s to store data.
- MongoDB stores documents (or) objects.
- Now-a-days, everyone works with objects (Python/Ruby/Java/etc.)
- And we need Databases to persist our objects. Then why not store objects directly?
- Embedded documents and arrays reduce need for joins. No Joins and No-multi document transactions.
Why use MongoDB?

- Simple queries
- RDBMS replacement for Web Applications.
- Easier and faster integration of data
- Semi-structured Content Management.
- Real-time Analytics & High-Speed Logging.
- Caching and High Scalability
- Not well suited for heavy and complex transactions systems.
The Basics

• A MongoDB instance may have zero or more databases

• A database may have zero or more ‘collections’.

• A collection may have zero or more ‘documents’.

• A document may have one or more ‘fields’.

• MongoDB ‘Indexes’ functions much like their RDBMS counterparts.
MongoDB vs. RDBMS

- Collection vs. table
- Document vs. row
- Field vs. column
- Collection isn't strict about what goes in it (it's schema-less)
MongoDB in Action...

```
C:\windows\system32\cmd.exe - c:\mongodb\bin\mongo

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\Truc Pham>C:\mongodb\bin\mongo
MongoDB shell version: 2.0.3
connecting to: test
> use learn
switched to db learn
> db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})
> db.unicorns.find()
{ "_id" : ObjectId("4f6599d0e7923e588728343f"), 
  "name" : "Aurora", 
  "gender" : "f",
  "weight" : 450 } 
> db.unicorns.insert({name: 'Leto', gender: 'm', home: 'Arrakeen', worm: false})
> db.unicorns.find()
{ "_id" : ObjectId("4f6599d0e7923e588728343f"), 
  "name" : "Aurora", 
  "gender" : "f",
  "weight" : 450 } 
{ "_id" : ObjectId("4f6599f6be7923e5887283440"), 
  "name" : "Leto", 
  "gender" : "m",
  "home" : "Arrakeen", 
  "worm" : false }
> db.unicorns.remove()
> db.unicorns.count()
0
```
connecting to: test
> use learn
switched to db learn
> db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0), loves: ...
["carrot", "grape"], weight: 450, gender: 'f', vampires: 43});
> db.unicorns.insert({name: 'Unicron', dob: new Date(1973, 1, 9, 22, 10), loves: ...
["energon", "redbull"], weight: 984, gender: 'm', vampires: 182});
> db.unicorns.insert({name: 'Roootoodles', dob: new Date(1979, 7, 18, 18, 44), ...
loves: ["apple"], weight: 575, gender: 'm', vampires: 99});
> db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57), loves: ...
["apple", "sugar"], weight: 421, gender: 'm', vampires: 2});
> db.unicorns.insert({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15), loves: ...
["grape", "carrot"], weight: 540, gender: 'f'});
> db.unicorns.find({gender: 'm', weight: {$gt: 700}})
{ "_id": ObjectId("4f66c211a3f7341b825b087"), "name": "Unicron", "dob": ISODate("1973-02-10T06:10:00Z"), "loves": ["energon", "redbull"], "weight": 984, "gender": "m", "vampires": 182 }
> db.unicorns.find({vampires: {$exists: false}})
{ "_id": ObjectId("4f66c23e4a3f7341b825b08b"), "name": "Nimue", "dob": ISODate("1999-12-21T00:15:00Z"), "loves": ["grape", "carrot"], "weight": 540, "gender": "f" }
> db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {loves: 'orange'}, { ...
weight: {$lt: 500}]}])
{ "_id": ObjectId("4f66c211a3f7341b825b087"), "name": "Aurora", "dob": ISODate("1991-01-24T21:00:00Z"), "loves": ["carrot", "grape"], "weight": 450, "gender": "f", "vampires": 43 }
Theory of noSQL: CAP

- Many nodes
- Nodes contain *replicas of partitions* of data
- Consistency
  - all replicas contain the same version of data
- Availability
  - system remains operational on failing nodes
- Partition tolerance
  - multiple entry points
  - system remains operational on system split

**CAP Theorem:** satisfying all three at the same time is impossible
Theory of noSQL: CAP (Brewer’s Theorem)

- Many nodes
- Nodes contain *replicas of partitions* of data
- Consistency
  - all replicas contain the same version of data
- Availability
  - system remains operational on failing nodes
- Partition tolerance
  - multiple entry points
  - system remains operational on system split

CAP Theorem: satisfying all three at the same time is impossible
ACID - BASE

- Atomicity
- Consistency
- Isolation
- Durability

- Basically Available (CP)
- Soft-state
- Eventually consistent (AP)
RDBS – ACID Model

• **Atomicity.** All of the operations in the transaction will complete, or none will.

• **Consistency.** The database will be in a consistent state when the transaction begins and ends.

• **Isolation.** The transaction will behave as if it is the only operation being performed upon the database.

• **Durability.** Upon completion of the transaction, the operation will not be reversed.
Non-RDBS – BASE Model

• **Basically Available:**
  – some parts of system remain available on failure

• **Soft-state:**
  – the information will expire unless it is refreshed
  – system will change state without user intervention due to eventual consistency

• **Eventually consistency:**
  – asynchronous propagation
  – consistency window
Mongoose for Data Modelling

• MongoDB’s flexibility about what it stores in documents is a great thing for the database.

• But most applications need some structure to their data.

• Mongoose provides “elegant MongoDB object modeling for Node.js”
  – (http://mongoosejs.com/).

• Data modeling, in the context of Mongoose and MongoDB is defining what data *can* be in a document, and what data *must* be in a document.
What Else Does Mongoose Offer?

- Mongoose adds an entire layer of features on top of MongoDB that are useful when building web applications.
- Mongoose makes it easier to manage the connections to your MongoDB database, as well as to save data and read data.
- Mongoose enables you to add data validation at the schema level, making sure that you only allow valid data to be saved in the database.
What is ExpressJS?

• Express is a relatively small framework that sits on top of Node.js’s web server functionality to simplify its APIs and add helpful new features.

• It makes it easier to organise your application’s functionality with middleware and routing; it adds helpful utilities to Node.js’s HTTP objects;

• It facilitates the rendering of dynamic HTML views;

• It defines an easily implemented extensibility standard.

• It provides *Middlewares* and *Routing*
What is Express?

• **Middlewares are different request handlers** such as Logging middleware, Authorization middleware, send the homepage and so on.

• **Routing** allows you to partition your application’s behavior by route.

• Express has some small features for rendering HTML.

• Some popular templating languages come with Express support, such as Pug

• Express doesn’t have any notion of a database, you can persist your application’s data however you choose: in files, in a relational SQL database, or MongoDB
Node.js without Express

• Building your application on top of Node.js without Express

Flow of a request through a Node.js Web Application
Node.js with Express

• What Express adds to Node.js

Flow of a request through a Node.js Web Application

• Rather than one large request handler function, Express makes you writing many smaller functions.
The core parts of Express

Express has just four major features: *middleware*, *routing*, *sub-applications*, and *conveniences*.

**Middleware**

- Node.js gives you one request handler function to work with. The request comes into your function and the response goes out of your function.

- The idea is pretty simple: rather than *one* monolithic request handler function, you call *several* request handler functions that each deal with a small chunk of the work.

- These smaller request handler functions are called *middleware functions*, or middleware.
Middleware-based Request Handling

- Notice that middleware sometimes continues on, but sometimes it responds to requests.

1. Request A comes in
2. Logging done, continue on
3. User is authorized, continue on.
4. Respond with secret info

1. Request B comes in
2. Logging done, continue on
3. User is not authorized, respond with error and do not continue.

“Send the secret info” middleware

COMP6205 - MEAN Stack 28
Routing

- Like middleware, it breaks the one monolithic request handler function into smaller pieces.

- Unlike middleware, these request handlers are executed conditionally, depending on what URL and HTTP method a client sends.

  - For example, you might build a web page with a homepage and a guestbook. When the user sends an HTTP GET to the homepage URL, Express should send the homepage. But when they visit the guestbook URL, it should send them the HTML for the guestbook, not for the homepage! And if they post a comment in the guestbook (with an HTTP POST to a particular URL), this should update the guestbook. Routing allows you to partition your application’s behavior by route.
Subapplications

- Express applications can often be pretty small, even fitting in just one file.
- As your applications get larger, though, you’ll want to break them up into multiple folders and files.
- Express is unopinionated about how you scale your app, but it provides one important feature that’s super helpful: subapplications.
- In Express lingo, these miniapplications are called routers.
Subapplications

An example diagram showing how a large application could be broken up into routers
Conveniences

• Express applications are made up of middleware and routes, both of which have you writing request handler functions, so you’ll be doing that a lot!

• To make these request handler functions easier to write, Express has added a bunch of niceties.

• For example in raw Node.js, if you want to write a request handler function that sends a JPEG file from a folder, that’s a fair bit of code.
  – In Express, that’s only one call to the `sendFile` method.
  – Express has a bunch of functionality for rendering HTML more easily;
Components of MEAN stack

• MongoDB
  • Document-based (JSON) NoSQL database powered by JavaScript

• ExpressJS
  • JavaScript back-end framework

• AngularJS
  • JavaScript front-end framework

• NodeJS
  • JavaScript run-time (C/C++ taps into Chrome V8 engine)
AngularJS

• AngularJS helps put together HTML based on the data provided.
  • It also immediately updates the HTML if the data changes,
  • and can also update the data if the HTML changes.
• This is known as two-way data binding, which we’ll now take a quick look at.
• Great client-side development framework
• Goolge-backed
• Bower helps manage client-side packages
More on Bower

- There are now more libraries and frameworks available for front-end development than ever before.
- It’s not uncommon to have five or more of these libraries involved in a single project. But keeping track of all these libraries and making sure they’re up-to-date can be tricky.

- Bower a package manager that makes it easy to manage all your application’s front-end dependencies.

https://bower.io/
One-way Data Binding

The template and model are compiled on the server before being sent to the browser.
Two-way Data Binding

- The model and the view are processed in the browser and bound together, each instantly updating the other.
Using AngularJS to Load New Pages

• AngularJS has been specifically designed for *Single-Page Application* (SPA) functionality.

• SPA runs everything inside the browser and never does a full page reload.

• What this means is that all application logic, data processing, user flow, and template delivery can be managed in the browser.
  – This approach can really reduce the amount of resources you need on your server
  – The user experience can also be better when using this approach.
  – Once the application is loaded there are fewer calls to be made to the server, reducing the potential of latency.
Are There any Downsides?

- AngularJS uses JavaScript to build the rendered HTML from templates and data, so if your browser doesn’t support JavaScript, or if there’s a bug in the code, then the site won’t run.
- Reliance on JavaScript to build the page also causes problems with search engines.
- JavaScript applications are very hard for search engines to crawl and index - (SEO Issues)
  - With SPA the only thing you get before JavaScript takes over is the template from the server.
  - There are ways to combat this issue but they come at a cost.
Are There any Downsides?

• Analytics and browser history
  – Analytics tools like Google Analytics rely heavily on entire new pages loading in the browser, initiated by a URL change.
  – You can add page load events to an SPA using the HTML5 history API;

• Speed of initial load
  – SPAs have a slower first page load than server-based applications.
  – This is because the first load has to bring down the framework and the application code before rendering the required view as HTML in the browser.
  – caching and lazy-loading can help with first page loading
SPA or Non-SPA?

Requirements for a blog engine

Blog entries

Characteristics:
- Content-rich
- Low interaction
- Fast first load
- Short user duration
- Public and shareable

Admin interface

Characteristics:
- Feature-rich
- High interaction
- Fast response to actions
- Long user duration
- Private
Choice of Architecture

- AngularJS SPA making use of a REST API built with MongoDB, Express, and Node.js

- An architecture for delivering HTML directly from the server
A Hybrid MEAN Stack Architecture

- A single REST API feeding two separate user-facing applications,

- Built using different parts of the MEAN stack to provide the most appropriate solution
Options for Building MEAN Stack Web Apps

- Options ranging from a Server-side Express and Node.js application to a full client-side AngularJS SPA
Web Architectures

1. A Single Server Architectures

Database → REST API → Application

2. Shared Server Approach

Database → REST API → Application

3. A Decoupled Architecture Using Three Servers

Database → REST API → Application

4. Clusters of Servers for Each Part of Application

Database → REST API → Application
NodeJS

- Node.js is a software platform that allows you to create your own web server and build web applications on top of it.
  - Node.js (often shortened to Node) is a JavaScript platform—a way to run JavaScript on the server side.
  - Runs on the Chrome “V8 engine”, written in C/C++
  - It comes with a great package manager called NPM
- It contains a built-in HTTP server library, meaning that you don’t need to run a separate web server program such as Apache or Internet Information Services (IIS).
The Idea Behind node.js....

- Node.js encourages an **asynchronous** coding style, making for faster code while **avoiding multithreaded** approach.

- It is based on **Non-Blocking I/O**
  
  - If no results are available at the moment of the call, the function will simply return a predefined constant, indicating that there is no data available to return at that moment.
  
- Ideal for applications that serve a lot of requests but don’t use/need lots of computational power per request.

- Not so ideal for heavy calculations, e.g. massive parallel computing.
Node.js Architecture

- V8, the JavaScript engine called V8, originally developed by Google for the Chrome browser.
  - This is one of the reasons why Node.js is so fast and efficient.
  - V8 is acclaimed for its revolutionary design, its speed, and for its efficient memory management.
Node.js Architecture – Cont.

- libuv represents the low-level I/O engine of Node.js.
- libuv is a C library, with the objective to make Node.js compatible with all the major platforms and normalize the non-blocking behaviour of the different types of resource.
- Bindings responsible for wrapping and exposing libuv and other low-level functionality to JavaScript.
- A core JavaScript library (called node-core) that implements the high-level Node.js API.
Blocking vs Non-Blocking......

• Example :: Read data from file and show data

![Diagram showing blocking and non-blocking I/O]

**Synchronous I/O**
- Thread waits during I/O operation
  - Thread
  - File I/O

**Asynchronous I/O**
- Thread DON'T wait during I/O operation
  - Thread
  - File I/O
Blocking

Blocking Wastes Cycles

- Read File
- Send HTTP Request
- SQL Query

- Waiting on File I/O...
- Waiting on Network Response...
- Waiting on DB...

Time
Non-Blocking

• Run entirely in a single thread
• Passes I/O requests to the event loop, along with callbacks
• Your code then: Goes to sleep, Uses no system resources
• Will be notified via callback when I/O is complete
Node.js App

- Node.js operates on a **single thread**, using **non-blocking I/O calls**, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching.

- Passes I/O requests in the event loop, along with callbacks

- Your code then:
  
  - Goes to sleep
  
  - Uses no system resources
  
  - Will be notified via callback when I/O is complete
Traditional Web Application Processing Model

- Web Applications typically follow “Multi-Threaded Request-Response” model. Simply we can call this model as Request/Response Model.

- Client sends HTTP request to the server, then the server does some processing based on clients request, prepare HTTP response and send it back to the client.

- This Request/Response model is Stateless.

- This model uses Multiple Threads to handle concurrent client requests.
• If more clients requests require Blocking IO Operations, then almost all threads are busy in preparing their responses. Then remaining clients Requests should wait for longer time.
Request/Response Model Processing Steps:

- Clients Send request to Web Server.
- Web Server is in infinite Loop and waiting for Client Incoming Requests
- Web Server receives those requests.
  - Web Server pickup one Client Request
  - Pickup one Thread from Thread pool (Web Server maintains a Limited Thread pool to provide services to the Client Requests.)
  - Assign this Thread to Client Request
  - This Thread will take care of reading Client request, processing Client request, performing any Blocking IO Operations (if required) and preparing Response
  - This Thread sends prepared response back to the Web Server
  - Web Server in-turn sends this response to the respective Client.
- Server waits in Infinite loop and performs all sub-steps as mentioned above for all clients. That means this model cannot use Threads.
Single Thread, Non-Blocking I/O
Single Thread, Non-Blocking I/O

1. The application generates a new I/O operation by submitting a request to the Event Demultiplexer. The application also specifies a handler, which will be invoked when the operation completes. Submitting a new request to the Event Demultiplexer is a non-blocking call and it immediately returns control to the application.

2. When a set of I/O operations completes, the Event Demultiplexer pushes the new events into the Event Queue.

3. At this point, the Event Loop iterates over the items of the Event Queue.
Single Thread, Non-Blocking I/O

4. For each event, the associated handler is invoked.

5. The handler, which is part of the application code, will give back control to the Event Loop when its execution completes (5a). However, new asynchronous operations might be requested during the execution of the handler (5b), causing new operations to be inserted in the Event Demultiplexer (1), before control is given back to the Event Loop.

6. When all the items in the Event Queue are processed, the loop will block again on the Event Demultiplexer which will then trigger another cycle when a new event is available.
Node Web Apps

Node.js Server

*Request*

*Request*

*Requests*

*Requests*

Event Loop

Delegate

POSIX Async Threads

Non-blocking IO

Thread Processing

Thread Waiting
Non-Blocking

- Read data from file

When read data completed, show data

- Do other tasks

```javascript
fs.readFile("test.txt", function(err, data) {
  console.log(data);
});
```
Blocking vs Non-Blocking Code

• Create a text file named `input.txt` with the following content:

  Tutorials Point is giving self learning content to teach the world in simple and easy way!!!!!

• Create a js file named `main.js` with the following code:

```javascript
var fs = require("fs");

var data = fs.readFileSync('input.txt');

code.log(data.toString());
code.log("Program Ended");
```
Blocking Code Output

• Now run the main.js to see the result

```
$ node main.js
```

• Programme Output.

Tutorials Point is giving self learning content to teach the world in simple and easy way!!!!!! Program Ended
Non-Blocking Version

• Update main.js to have the following code:

```javascript
var fs = require("fs");

fs.readFile('input.txt', function (err, data) {
  if (err) return console.error(err);
  console.log(data.toString());
});

console.log("Program Ended");
```

• Now run the main.js to see the result

• Program output

```bash
$ node main.js
Program Ended
Tutorials Point is giving self learning content to teach the world in simple and easy way!!!!!
```
MEAN/MERN stack

- **Node.js:**
  - [https://nodejs.org/en/](https://nodejs.org/en/)
- **mongoDB:**
  - [https://www.mongodb.com/](https://www.mongodb.com/)
- **Express:**
  - [https://expressjs.com/](https://expressjs.com/)
- **AngularJS:**
  - [https://angularjs.org/](https://angularjs.org/)
- **React (A JavaScript library for building user interfaces):**
  - [https://reactjs.org/](https://reactjs.org/)
  - Can replace AngularJS
Node.js Modules

- [https://npmjs.org/](https://npmjs.org/)

- npm is a package manager that gets installed when you install Node, which gives you the ability to download Node modules or packages to extend the functionality of your application.

- In every Node application there should be a file in the root folder of the application called `package.json`. This file can contain various metadata about a project, including the packages that it depends on to run.

- Install module: `npm install <module name>`

  ```bash
  $ npm install --save package-name
  ```
Defining packages & dependencies with package.json

• In every Node application there should be a file in the root folder of the application called `package.json`.

```json
{
  "name": "application-name",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.9.0",
    "body-parser": "~1.8.1",
    "cookie-parser": "~1.3.3",
    "morgan": "~1.3.0",
    "serve-favicon": "~2.1.3",
    "debug": "~2.0.0",
    "jade": "~1.6.0"
  }
}
```
Using module

- var http = require('http');
- var fs = require('fs');
- var express = require('express');

```javascript
var url = require("url");
var parsedURL = url.parse("http://www.example.com/profile?name=barry");

console.log(parsedURL.protocol); // "http:
console.log(parsedURL.host); // "www.example.com"
console.log(parsedURL.query); // "name=barry"
```
Hosting with Heroku

- Hosting Node.js applications can be complicated, but it doesn’t have to be.
  - Many traditional shared hosting providers haven’t kept up with the interest in Node.js.

- To run a Node.js application successfully you either need
  - a server that has been configured with that in mind,
  - or you can use a PaaS provider that’s aimed specifically at hosting Node.js.

- Heroku is one of the leading hosts for Node.js applications (www.heroku.com)
  - Once everything is set up you can publish your application to a live environment using a single command:

  ```
  $ git push heroku master
  ```
References

• Getting MEAN With Mongo, Express, Angular, and Node Book by Simon Holmes

• Express in Action, Writing, building, and testing Node.js applications