Overview

Demo following technologies:

- JSON
- Node Package Manager `npm`
- Node modules.
- Very brief introduction to asynchronous programming using `async` and `await`.
- Mongo db

JSON

JavaScript Object Notation.
Inductive definition for JSON values:

Primitives `null`, `true`, `false`, numbers and "-quoted strings. Minimal set of escape sequences in strings.

Sequences Comma-**separated** JSON values within `[ ]`.

Maps Comma-separated key-value pairs within `{ }`. Keys must be JSON strings and values are JSON values.

Though JSON bears a superficial resemblance to JavaScript object literals, it is a different notation.

Example

JSON Evaluation

- Widely popular for transferring structured data between heterogeneous systems.
- Preferred over XML for structured **data** (XML is good for structured **documents**).
- Not suitable as a configuration format as no comments allowed. (YAML is a better format).
- In JavaScript, built-in `JSON` object provides `stringify()` and `parse()` methods to convert JavaScript objects to / from a string.
- Some JSON libraries allow comments (and other features like terminating commas) as syntax extensions, but not as per JSON standard.

Node Package Manager

- Manages dependencies between packages or modules.
- Local packages: dependencies of your project.
- Global packages: use for CLI tools.
- `package.json` describes project dependencies and `package-lock.json` serves to lock-down dependency versions.
- By default, packages are installed in `node_modules` directory of current directory.
- Usually `package.json` and `package-lock.json` are checked into version control but not the `node_modules` directory. To run a project after checking it out from version control it is usually enough to simply run `npm install`

Semantic Versioning

Semantic Versioning attempts to avoid *dependency hell*. It uses a 3 part version number: *M.m.r* where each part is a integer without leading zeros.

Revision Number *r* Incremented for bug fixes.

Minor Version *m* Incremented for added functionality which is backward compatible.

Major Version *M* Incremented for incompatible changes which are not backward compatible.

Node Modules

- Modules provide encapsulation; Entities like variables, functions, classes are inaccessible outside a module unless explicitly exported by the module.
- A nodejs module is either a JavaScript file, or a directory with either a `package.json` or `index.js` file in it.
- A file can export information by assigning to `module.exports`. The value which is assigned can be any JavaScript value; common values exported are either a single function or class, or a JavaScript object mapping names to JavaScript entities.
- When a module is `require`'d, the return value of the `require()` is simply the value which was assigned to `module.exports`.

Examples of Using require()

Assume ./myModule.js exports using

```
function f() { ... }
class C { ... }
const VAL = ...;
module.exports = { f: f, C: C, VAL: VAL };
```

We can require entire module using:

```
const myModule = require('./myModule');
//code can use myModule.f, myModule.C, myModule.VAL
```

We can require only part of exports using destructuring:

```
const { VAL: myVal } = require('./myModule');
```

Require Paths

- If a module is required using a relative or absolute path, then the folder or file at that path is loaded (trying extensions `.js` or `.json`).
- If a module is required without specifying a path, then it is looked for in the user's directory starting with the current directory as well as in global installation directories.

```
> module.paths
[ '/home/umrigar/tmp/repl/node_modules',
'/home/umrigar/tmp/node_modules',
'/home/umrigar/node_modules',
'/home/node_modules',
'/node_modules',
'/home/umrigar/.node_modules',
'/home/umrigar/.node_libraries',
'/usr/lib/nodejs' ]
>
```

MongoDb

- One of many nosql databases. No rigid relations need to be predefined.
- Allows storing and querying json documents.
- Provides basic **Create-Read-Update-Delete** (CRUD) repertoire.

Mongo Crud

Create	`insertOne()` and `insertMany()`.
Read	`find()` returns a `Cursor`. Can grab all using `toArray()`.
Update	`updateOne()` and `updateMany()`. Also, `findOneAndUpdate()` and `findOneAndReplace()`.
Delete	`deleteOne()` and `deleteMany()`.

All functions require a callback, but will return a `Promise` if called without a callback.

User Store Features

- Store user-info objects.
- No schema for user-info objects, except that each object **must** have a `id` property.
- Have `id` property default to email set in global git configuration for current user.
- Basic CRUD functionality.

Log

```
$ ./index.js read lisa
NOT_FOUND: user(s) {"id":"lisa"} not found
$ ./index.js create simpsons.json
$ ./index.js create simpsons.json
EXISTS: user(s) bart, marge, lisa, homer already exist
```

Log Continued

```
$ ./index.js read homer lisa
[
  {
    "id": "homer",
    "firstName": "Homer",
    "lastName": "Simpson",
    "email": "chunkylover53@aol.com"
  },
  {
    "id": "lisa",
    "firstName": "Lisa",
    "lastName": "Simpson",
    "birthDate": "1982-05-09",
    "email": "smartgirl63_\\@yahoo.com"
  }
]
```

Log Continued

```
$ ./index.js delete lisa
$ ./index.js read lisa
NOT_FOUND: user(s) {"id":"lisa"} not found
$ ./index.js update homer birthdate=1953-03-31
$ ./index.js read homer
[
  {
    "id": "homer",
    "firstName": "Homer",
    "lastName": "Simpson",
    "email": "chunkylover53@aol.com",
    "birthdate": "1953-03-31"
  }
]
```

Log Continued

```
$ ./index.js create   #default id set to git email
$ ./index.js read umrigar@binghamton.edu
[
  {
    "id": "umrigar@binghamton.edu"
  }
]
$ ./index.js update umrigar@binghamton.edu \
                    name='zerksis umrigar'
$ ./index.js read umrigar@binghamton.edu
[
  {
    "id": "umrigar@binghamton.edu",
    "name": "zerksis umrigar"
  }
]
$
```

Initializing Project

```
$ npm init -y
...
$ npm install --save mongodb
npm notice created a lockfile as package-lock.json...
...
added 6 packages in 6.074s
$ ls -a
.   ..   .gitignore   node_modules   package.json
package-lock.json
$
```

async and await

- Friendlier syntax layered over more basic asynchronous facilities.
- Allows calling asynchronous functions in a synchronous style.
- Declare asynchronous functions using the `async` keyword.
- Call asynchronous functions using the `await` keyword.
- `await` is recognized only within the body of a function which has been declared `async`.
- `async` / `await` is a new addition to JavaScript. Older nodejs API's need to be promisified using util.promisify().

Implementation

index.js Wrapper which dispatches to command-line handling.

user-store-cli.js Command-line handling.

user-store.js Implementation of db operations.

Mongo Shell Log

Allows interacting with mongo db. Following log assumes that collection `userInfos` in db `users` is loaded with simpsons data.

```
$ mongo
MongoDB shell version: 3.2.11
...
> use users
switched to db users
> db.userInfos.find({})
{ "_id" : "bart", "id" : "bart", ... }
{ "_id" : "marge", "id" : "marge", ... }
{ "_id" : "lisa", "id" : "lisa", ... }
{ "_id" : "homer", "id" : "homer", ... }
> db.userInfos.find({"firstName": "Bart"})
{ "_id" : "bart", "id" : "bart", ... }
> db.userInfos.find({}).length()
4
```

Mongo Shell Log Continued

```
> db.userInfos.remove({"firstName": "Bart"})
WriteResult({ "nRemoved" : 1 })
> db.userInfos.find({}).length()
3
> db.userInfos.remove({})
WriteResult({ "nRemoved" : 3 })
> db.userInfos.find({}).length()
0
```