1

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. Mongoose has a very rich API that handles many complex operations supported by MongoDB.
MongoDB is a de-normalized NoSQL database. This makes it inherently schema-less as documents have varying sets of fields with different data types. Mongoose defines a schema for our data models so our documents follow a specific structure with pre-defined data types.

Create Record

```
let EmailModel = require('./email')

let msg = new EmailModel({
  email: 'ADA.LOVELACE@GMAIL.COM'
})
msg.save()
   .then(doc => {
     console.log(doc)
   })
   .catch(err => {
     console.error(err)
   })
```

Read Record

```
EmailModel
  .find({
    email: 'ada.lovelace@gmail.com'   // search query
  })
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Update Record

```
EmailModel
  .findOneAndUpdate(
    {
      email: 'ada.lovelace@gmail.com'  // search query
    },
    {
      email: 'theoutlander@live.com'   // field:values to update
    },
    {
      new: true,                       // return updated doc
      runValidators: true              // validate before update
    })
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Delete Record

```
EmailModel
  .findOneAndRemove({
    email: 'theoutlander@live.com'
  })
  .then(response => {
    console.log(response)
  })
  .catch(err => {
    console.error(err)
  })
```

2

The **POST** method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database. In other words, POST is used to append a resource to an existing collection.

The **PUT** method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource then the origin server can create the resource with that URI.

The POST is open-ended enough that it can be idempotent if we wish, but HTTP does not require it to be. So we have not broken with HTTP, just missed the opportunity to use the more germane PUT method. That is why, when we used the POST to update employee data in the Employees sample AJAX application, we had no issue & everything went well.

If we replace the POST with the PUT then the origin server creates resource with that URI if it doesn't exist otherwise, the enclosed entity would be considered as its modified version. But using PUT for a non-idempotent part of the API, would be more problematic.

3

The difference between the PUT and PATCH requests is reflected in the way the server processes the enclosed entity to modify the resource identified by the Request-URI.

In a **PUT** request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced.

With **PATCH**, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version. The PATCH method affects the resource identified by the Request-URI, and it also MAY have side effects on other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH.

We can use whatever format we want as [description of changes], as far as its semantics is well-defined. That is why using PATCH to send updated values only is not suitable.

Therefore, the PUT should be used for partial updates like in the case of updating name in an online university application form.

	REACT	ANGULARJS
TYPE	Frontend-library	Framework
TEMPLATE	JSX + JS (ES5/ES6)	HTML + TypeScript
DATA BINDING	Uni-directional	Bi-directional
DOM	Virtual DOM	Regular DOM
ABSTRACTION	Medium	Strong
LEARNING CURVE	Medium	High
RELEASED	2013	2016
RENDEREING	Server-side	Client-side
DEPENDENCIES	Requires additional tools	Manages automatically
TECHNOLOGY	JavaScript library	Full-fledged MVC framework

React

Advantages:

- Virtual DOM only updates the part that is required by seeing the differences between the previous and current HTML version which influences performance & as well as user-experience.
- React handles memory-management efficiently (virtual DOM)
- Dynamically typed language is readable and easier to write (declarative code)
- Permission to reuse React components significantly saves time
- One-direction data flow provides a stable code

Disadvantages:

- It is only a view layer, we have still to plug our code for Ajax requests, events and so on.
- The library itself is quite large.
- Integrating React.js into a traditional MVC framework such as rails would require some configuration (i.e., substituting erb with React.js).

AngularJS

Advantages:

- AngularJS has many standard directives, such as ng-bind or ng-app, but we can create own directives as well which is a powerful way to work with the DOM.
- AngularJS uses a basic Object Oriented Programming (OOP) pattern called dependency injection, meaning we write dependencies in a separate file. It's inconvenient to create a dependency directly in an object.
- AngularJS is a comprehensive solution for rapid front-end development. It does not need any other plugins or frameworks. Moreover, there are a range of other features that include Restful actions, data building, enterprise-level testing, etc.

Disadvantages:

- Regular DOM updates the whole tree structure of HTML tags until it reaches the birth date which influences performance & as well as user-experience.
- Learning a statically-typed language may be a challenge, especially if one has only been working with dynamically typed languages
- Statically typed language require more time due to fixing typo errors

5

Vue is a **progressive framework** for building user interfaces. It describes itself as an "*Intuitive, Fast and Composable MVVM for building interactive interfaces*." Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.

Comparison of Vue with React & Angular

- React and Vue both excel at handling dumb components: small, stateless functions that receive an input and return elements as output.
- Angular relies on TypeScript. React focuses on the use of Javascript ES6. Vue uses Javascript ES5 or ES6.
- We can use Flow to enable type-checking within React. It's a static type-checker developed by Facebook for JavaScript. Flow can also be integrated into VueJS.

- React and Vue, on the other hand, are universally flexible. Their libraries can be paired to all kinds of packages.
- Vue seems to be the cleanest and lightest of the three frameworks.
- One of the big cons of React vs. Vue is the problem of splitting components into smaller components because of the JSX restrictions.
- Vue can make use of Redux — but it offers Vuex as its own solution.
- Vue supports both one-way-binding and two-way-binding (one-way by default).
- React and Vue give us more control to size an application by selecting only the things which are really necessary. They offer more flexibility to shift from an SPA to microservices using parts of a former application. Angular work best for SPA, as it is probably too bloated to be used for microservices.
- The gzipped file size of Vue is 23K, compared to 143K for Angular and 43k for React.
- Vue lacks testing guidance.
- For React there is next.js , Vue has nuxt.js, and Angular has Angular Universal.
- Vue is pretty easy to learn. Companies switch to Vue because it seems to be much easier for junior developers. With Vue, the gap between junior and senior developers shrinks, and they can collaborate more easily and with fewer bugs, problems and time to develop.
- Vue looks more like plain Javascript while also introducing some new ideas: components, an event-driven-model, and one-way data flow. It also has a small footprint.
- With Vue, we can do the things the old-Javascript-fashioned way.

6

1. Architecture

AngularJS

The architecture of AngularJS is based on model-view-controller (MVC) design. The model is the central component that expresses the application's behavior and manages its data, logic, and rules. The *view* generates an output based on the information in the *model*. The *controller* accepts input, converts it into commands and sends the commands to the *model* and the *view*.

Angular

In Angular 2, controllers and $scope were replaced by components and directives. Components are directives with a template. They deal with a view of the application and logic on the page. There are two kinds of directives in Angular 2. These are structural directives that alter the layout of the DOM by removing and replacing its elements, and attributive directives that change the behavior or appearance of a DOM element.

In Angular 4, the structural derivatives ngIf and ngFor have been improved, and we can use if/else design syntax in our templates

2. Language

AngularJS

AngularJS is written in JavaScript.

Angular

Angular uses Microsoft's TypeScript language, which is a superset of ECMAScript 6 (ES6). This has the combined advantages of the TypeScript features, like type declarations, and the benefits of ES6, like iterators and lambdas.

Angular 4 is compatible with the most recent versions of TypeScript that have powerful type checking and object-oriented features.

3. Expression Syntax

AngularJS

To bind an image/property or an event with AngularJS, we have to remember the right ngdirective.

Angular

Angular focuses on "( )" for event binding and "[ ]" for property binding.

4. Mobile Support

AngularJS was not built with mobile support in mind, but Angular 2 and 4 both feature mobile support.

5. Routing

AngularJS uses $routeprovider.when() to configure routing while Angular uses @RouteConfig{(…)}.

Performance

AngularJS was originally developed for designers, not developers. Although there were a few evolutionary improvements in its design, they were not enough to fulfill developer requirements. The later versions, Angular 2 and Angular 4, have been upgraded to provide an overall improvement in performance, especially in speed and dependency injection.

1. Speed

By providing features like 2-way binding, AngularJS reduced the development effort and time. However, by creating more processing on the client side, page load was taking

considerable time. Angular2 provides a better structure to more easily create and maintain big applications and a better change detection mechanism. Angular 4 is the fastest version yet.

2. Dependency injection

Angular implements unidirectional tree-based change detection and uses Hierarchical Dependency Injection system. This significantly boosts performance for the framework.

7

Code linting is a type of static analysis that is frequently used to find problematic patterns or code that doesn't adhere to certain style guidelines. As it is used for static code analysis, therefore part of white-box testing.

JSLint for checking if JavaScript source code complies with coding rules. It is provided primarily as a web application through jslint.com. It helps to program in that better language and to avoid most of the slop. JSLint will reject programs that browsers will accept because It is concerned with the quality of our code and browsers are not.

Example:

Some of ES6's features are good, so JSLint will recognize the good parts of ES6. Currently, these features are recognized:

- The ... *ellipsis* marker in parameter lists and argument lists, replacing the arguments object for variadic functions.
- The let statement, which is like the var statement except that it respects block scope. We may use let or var but not both.
- The const statement is like the let statement except that it disallows the use of assignment on the variable, although if the value of the variable is mutable, it can still be mutated. const is preferred to let.
- Destructuring of arrays and objects is allowed in parameter lists and on the left side of let, and const, but not varor assignment statements, and not deep destructuring or eliding.
- Enhanced object literals, providing shorter forms for function declaration and properties that are initialized by variables with the same name.
- The fat arrow => *fart* functions.
- The simplest forms of import and export.
- `Megastring` literals, but not nested `megastring` literals.

- New global functions, such as Map, Set, WeakMap, and WeakSet.
- 0b- and 0o- number literals.

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs. In many ways, it is similar to JSLint and JSHint with a few exceptions:

- ESLint uses Espree for JavaScript parsing.
- ESLint uses an AST to evaluate patterns in code.
- ESLint is completely pluggable, every single rule is a plugin and we can add more at runtime.

Examples:

1)

Let's take a look at this example:

```
function sum (a, b) {
return a + c;
}
```

The improper coding of this function is going to bug out. Linting will help to catch this code before it causes a problem.

2)

ESLint has a configuration hierarchy.

- **Within Project** – We can specify project-wide linting rules in the .eslintrc.* file in the project root. These rules apply to all *non-excluded* files in the directory.
- **Within Directory** – We can place a .eslintrc.* file in any directory to specify directory-wide rules. For example, we can add .eslintrc.* to a global testing folder to allow undefined in tests.
- **Within File** – We can disable or change rules in a file by using an eslint comment.
- Disable all rules in file – /* eslint-disable */.
- Disable specific rules in a file (comma separated) – /* eslint-disable no-console,no-alert */.
- Tweak a rule in a file – /* eslint no-undef:1 */.

AJAX can be preferred as a way to communicate (send request and get responses) with the server asynchronously i-e, without refreshing the page even AngularJS also uses AJAX to perform CRUD operations

But AngularJS is a full-fledged, front end MVC framework which does a lot more. It extends $http module with a lot of neat features such as 2-way data binding, templating, filters and directives etc. So in this case, AJAX is insufficient & we need to prefer AngularJS over it.