MongoDB

MongoDB is an open-source document database, and leading NoSQL database. MongoDB is written in C++ and is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

* **Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

* **Collection**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

* **Document**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

# Advantages of MongoDB over RDBMS

* Schema less: MongoDB is document database in which one collection holds different different documents. Number of fields, content and size of the document can be differ from one document to another.
* Structure of a single object is clear
* No complex joins
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL
* Tuning
* Ease of scale-out: MongoDB is easy to scale
* Conversion / mapping of application objects to database objects not needed
* Uses internal memory for storing the (windowed) working set, enabling faster access of data

# Why should use MongoDB

* Document Oriented Storage : Data is stored in the form of JSON style documents
* Index on any attribute
* Replication & High Availability
* Auto-Sharding
* Rich Queries
* Fast In-Place Updates
* Professional Support By MongoDB

# Where should use MongoDB?

* Big Data
* Content Management and Delivery
* Mobile and Social Infrastructure
* User Data Management
* Data Hub

Angular 2

* **package.json** lists packages the Wireup app depends on and defines some useful scripts.
  + **npm start**- runs the compiler and a server at the same time, both in "watch mode"
  + **npm run tsc**- runs the TypeScript compiler once
  + **npm run tsc:w** - runs the TypeScript compiler in watch mode; the process keeps running, awaiting changes to TypeScript files and recompiling when it sees them
  + **npm run lite** - runs the lite-server, a light-weight, static file server with excellent support for Angular apps that use routing
  + **npm run typings** - runs the typings tool separately
  + **npm run postinstall** - called by npm automatically after it successfully completes package installation. This script installs the TypeScript definition files defined in typings.json
* **tsconfig.json** is the TypeScript compiler configuration file.
* **typings.json**identifies TypeScript definition files.
* **systemjs.config.js**, the SystemJS configuration file.
* **AppComponent** Every Angular app has at least one root component, conventionally named AppComponent, that hosts the client user experience. Components are the basic building blocks of Angular applications. A component controls a portion of the screen — a view — through its associated template
  + A **@Component** decorator that tells Angular what template to use and how to create the component. Component is a decorator function that takes a metadata object as argument. We apply this function to the component class by prefixing the function with the @ symbol and invoking it with a metadata object, just above the class.
  + **A component class** that controls the appearance and behavior of a view through its template.
  + **Import** Angular apps are modular. They consist of many files each dedicated to a purpose. Angular itself is modular. It is a collection of library modules each made up of several, related features that we'll use to build our application.When we need something from a module or library, we import it. Here we import the Angular 2 core so that our component code can have access to the @Component decorator.
  + The **selector**specifies a simple CSS selector for an HTML element that represents the component. The element for this component is named wire-up. Angular creates and displays an instance of our AppComponent wherever it encounters a wire-up element in the host HTML.
  + The **template** specifies the component's companion template, written in an enhanced form of HTML that tells Angular how to render this component's view.
* **AppModule**
  + We compose Angular apps into closely related blocks of functionality with **Angular Modules**. Every app requires at least one module, the root module, that we call AppModule by convention.
  + **imports**- the other modules that export material we need in this module. Almost every application'sroot module should import the BrowserModule.
  + **declarations** - components and directives that belong to this module.
  + **bootstrap**- identifies the root component that Angular should bootstrap when it starts the application.
* **Main.ts**
  + We import the platformBrowserDynamic function from @angular/platform-browser-dynamic, not @angular/core. Bootstrapping isn't core because there isn't a single way to bootstrap the app. True, most applications that run in a browser call the bootstrap function from this library. But it is possible to load a module in a different environment. We might load it on a mobile device with Apache Cordova or NativeScript. We might wish to render the first page of our application on the server to improve launch performance or facilitate SEO. These targets require a different kind of bootstrap function that we'd import from a different library. main.ts, app module and the app component files are tiny. We'd rather demonstrate the proper way to structure an Angular application. App bootstrapping is a separate concern from creating a module or presenting a view.
* **Index.html** file defines the web page that hosts the application. The noteworthy sections of HTML are:
  + The JavaScript libraries
  + Configuration file for SystemJS, and a script where we import and run the app module which refers to the main file that we just wrote.
  + The <wire-app> tag in the <body> which is where our app lives
* **Polyfill(s) for older browsers** we begin with core-js's ES2015/ES6 shim which monkey patches the global context (window) with essential features of ES2015 (ES6). Next are the polyfills for Angular2, zone.js and reflect-metadata
* **SystemJS** to load application and library modules