FULL STACK LAB FILE

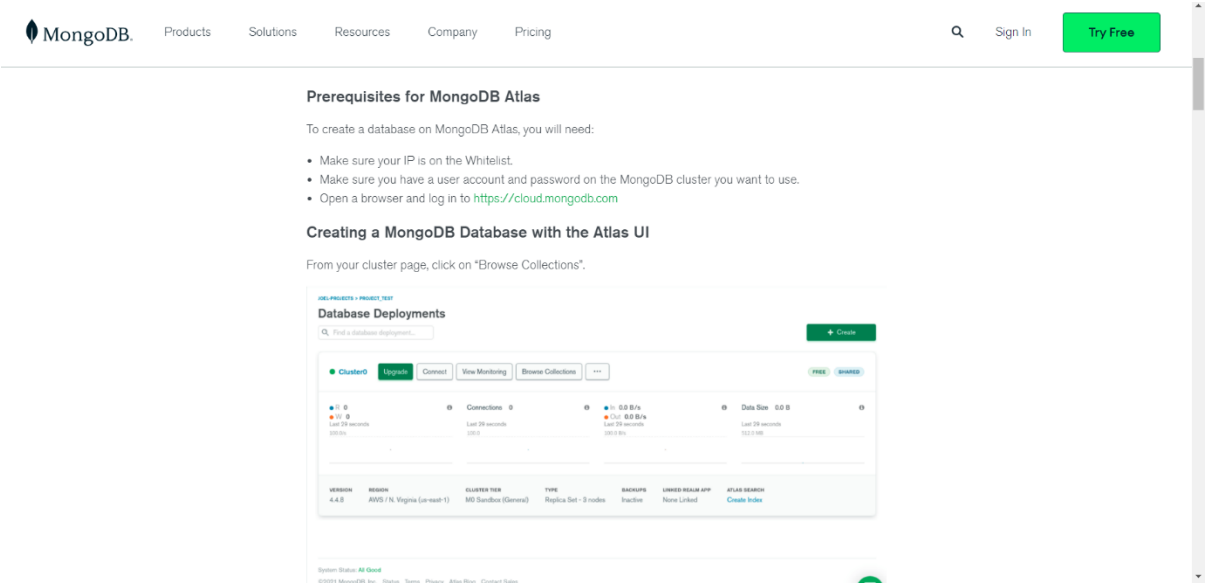
Installing and Managing MongoDB:-

- A Step-by-Step Guide to MongoDB Installation on Windows. Navigate to the official MongoDB website . Under the Software section, click on the Community server version.
- MongoDB Installation. You can find the downloaded file in the downloads directory. You can follow the steps mentioned there and install the software.
- Create an Environment Variable. Its best practice to create an environment variable for the executable file so that you dont have to change the directory structure every time you want.
- Execute the Mongo App. After creating an environment path, you can open the command prompt and just type in mongo and press enter.
- Verify the Setup. To verify if it did the setup correctly, type in the command show DBS.

Three most popular services to help you manage your MongoDB database:

- MongoDB Atlas makes running MongoDB in the public cloud easier than ever with fully managed database clusters, automatic...
- Use MongoDB Cloud Manager to run on-premise MongoDB infrastructure with professional-grade management, monitoring, and...
- MongoDB Compass is the desktop interface to MongoDB. Connect Compass to your database, whether it's on your laptop or in the cloud, and open up a whole world of insight into your database.

Create & Manage Database:-



Create & Manage collections:-

Basic syntax of **createCollection()** method without options is as follows −

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

You can check the created collection by using the command **show collections**.

```
>show collections
mycollection
system.indexes
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.tutorialspoint.insert({"name" : "tutorialspoint"}),
WriteResult({ "nInserted" : 1 })
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

Implementation of Modeling:-

Data Model Design:-

Effective data models support your application needs. The key consideration for the structure of your documents is the decision to embed or to use references.

Embedded Data Models:-

With MongoDB, you may embed related data in a single structure or document. These schema are generally known as "denormalized" models, and take advantage of MongoDB's rich documents. Consider the following diagram:

Embedded data models allow applications to store related pieces of information in the same database record. As a result, applications may need to issue fewer queries and updates to complete common operations.

In general, use embedded data models when:

you have "contains" relationships between entities. See Model One-to-One Relationships with Embedded Documents.

you have one-to-many relationships between entities. In these relationships the "many" or child documents always appear with or are viewed in the context of the "one" or parent documents. See Model One-to-Many Relationships with Embedded Documents.

In general, embedding provides better performance for read operations, as well as the ability to request and retrieve related data in a single database operation. Embedded data models make it possible to update related data in a single atomic write operation.

To access data within embedded documents, use dot notation to "reach into" the embedded documents. See query for data in arrays and query data in embedded documents for more examples on accessing data in arrays and embedded documents.

Embedded Data Model and Document Size Limit

Documents in MongoDB must be smaller than the maximum BSON document size.

For bulk binary data, consider GridFS.

Normalized Data Models:-

Normalized data models describe relationships using references between documents.

In general, use normalized data models:

when embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.

to represent more complex many-to-many relationships.

to model large hierarchical data sets.

Create your first AngularJS application in Visual Studio:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<script>
var app = angular.module("myShoppingList", []);
app.controller("myCtrl", function($scope) {
    $scope.products = ["Milk", "Bread", "Cheese"];
});
</script>

<div ng-app="myShoppingList" ng-controller="myCtrl">
  <ul>
    <li ng-repeat="x in products">{{x}}</li>
  </ul>
</div>

<p>So far we have made an HTML list based on the items of an array.</p>

</body>
</html>
```

- Milk
- Bread
- Cheese

So far we have made an HTML list based on the items of an array.

Implementation AngularJS Expressions:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<div ng-app>
<p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

My first expression: 10

Implementation AngularJS Modules:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>
<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.firstName = "Ajeet";
    $scope.lastName = "Maurya";
});
</script>
</body>
</html>
```

Ajeet Maurya

Implementation AngularJS Events:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1 ng-mousemove="count = count + 1">Mouse Over Me!</h1>

<h2>{{ count }}</h2>

</div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.count = 0;
});
</script>

</body>
</html>
```

Mouse Over Me!

40

Implementation AngularJS Filters & Services:-

Filter:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | uppercase }}</p>

</div>

<script>
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "John",
    $scope.lastName = "Doe"
});
</script>

</body>
</html>
```

The name is DOE

Service:-

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
<p>The url of this page is:</p>
<h3>{{myUrl}}</h3>
</div>

<p>This example uses the built-in $location service to get the absolute url of the page.
</p>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $location) {
    $scope.myUrl = $location.absUrl();
});
</script>

</body>
</html>
```

Implementing React Components:-

React lets you define components as classes or functions. Components defined as classes currently provide more features which are described in detail on this page. To define a React component class, you need to extend `React.Component`:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```