10a) Course Name: Angular JS
Module Name: Routing Basics, Router Links
Create multiple components and add routing to provide navigation between them.
Aim: Create multiple components and add routing to provide navigation between them.
Description:
Routing means navigation between multiple views on a single page.
Routing allows to express some aspects of the application's state in the URL. The full application can be built without changing the URL.
Routing allows to:

- Navigate between the views
- Create modular applications

ng generate component dashboard
ng generate component bookDetail
ng g c PageNotFound

Run Backend code parallel
And also install service book and book.ts and books-data in book component

Program:
dashboard.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  books: Book[] = [];
  constructor(
    private router: Router,
    private bookService: BookService) { }
  ngOnInit(): void {
    this.bookService.getBooks()
      .subscribe({next:books => this.books = books.slice(1,
5)});
  }

  gotoDetail(book: Book): void {
    this.router.navigate(['/detail', book.id]);
  }
}
```

dashboard.component.html:

```
<h3>Top Books</h3>
<div class="grid grid-pad">
```

```
  <div *ngFor="let book of books" (click)="gotoDetail(book)"
class="col-1-4">
    <div class="module book">
      <h4>{{ book.name }}</h4>
    </div>
  </div>
</div>
```

dashboard.component.css:

```
[class*="col-"] {
  float: left;
}
*,
*:after,
*:before {
  -webkit-box-sizing: border-box;
  -moz-box-sizing: border-box;
  box-sizing: border-box;
}
h3 {
  text-align: center;
  color:#607d8b;
  margin-bottom: 0;
}
[class*="col-"] {
  padding-right: 20px;
  padding-bottom: 20px;
}
[class*="col-"]:last-of-type {
  padding-right: 0;
}
.grid {
  margin: 0;
}
.col-1-4 {
  width: 25%;
}
.module {
  padding: 20px;
  text-align: center;
  color: #eee;
  max-height: 120px;
  min-width: 120px;
  background-color: green;
  border-radius: 2px;
}
h4 {
  position: relative;
}
.module:hover {
```

```
  background-color: #eee;
  cursor: pointer;
  color: #607d8b;
}
.grid-pad {
  padding: 10px 0;
}
.grid-pad > [class*="col-"]:last-of-type {
  padding-right: 20px;
}
@media (max-width: 600px) {
  .module {
    font-size: 10px;
    max-height: 75px;
  }
}
@media (max-width: 1024px) {
  .grid {
    margin: 0;
  }
  .module {
    min-width: 60px;
  }
}
```

book.service.ts:

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse, HttpHeaders,
HttpResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { catchError, tap, map} from 'rxjs/operators';
import { Book } from './book';
@Injectable({
    providedIn:'root'
})
export class BookService {
  booksUrl = 'http://localhost:3020/bookList';
  private txtUrl = './assets/sample.txt';
  constructor(private http: HttpClient) { }

  getBooks(): Observable<Book[]> {
    return this.http.get<any>(this.booksUrl,
{observe:'response'}).pipe(
      tap((data: any) => console.log('Data Fetched:' +
JSON.stringify(data))),
      map((data: any) => data.body),
      catchError(this.handleError));
  }

  getBook(id: any) {
```

```
    return this.getBooks().pipe(
      map((books) => books.find((book) => book.id == id))
    );
  }
  addBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type':
'application/json' });
    return this.http.post('http://localhost:3020/addBook',
book, { headers: options }).pipe(
      catchError(this.handleError));
  }
  updateBook(book: Book): Observable<any> {
    const options = new HttpHeaders({ 'Content-Type':
'application/json' });
    return this.http.put<any>('http://localhost:3020/update',
book, { headers: options }).pipe(
      tap((_: any) => console.log(`updated hero
id=${book.id}`)),
      catchError(this.handleError)
    );
  }
  deleteBook(bookId: number): Observable<any> {
    const url = `${this.booksUrl}/${bookId}`;
    return this.http.delete(url).pipe(
      catchError(this.handleError));
  }
  private handleError(err: HttpErrorResponse): Observable<any>
{
    let errMsg = '';
    if (err.error instanceof Error) {
      // A client-side or network error occurred. Handle it
accordingly.
      console.log('An error occurred:', err.error.message);
      errMsg = err.error.message;
    } else {
      // The backend returned an unsuccessful response code.
      // The response body may contain clues as to what went
wrong,
      console.log(`Backend returned code ${err.status}`);
      errMsg = err.error.status;
    }
    return throwError(()=>errMsg);
  }
}
```

book-detail.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Book } from '../book/book';
```

```
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
  book!: Book;
  error!: any;
  constructor(
    private bookService: BookService,
    private route: ActivatedRoute
  ) { }
  ngOnInit() {
      this.route.paramMap.subscribe(params => {

this.bookService.getBook(params.get('id')).subscribe((book) =>
{
        this.book = book ?? this.book;
      });
    });
  }
  goBack() {
    window.history.back();
  }
}
```

book-detail.component.html:

```
<div *ngIf="book">
  <h2>{{ book.name }} details!</h2>
  <div><label>Id: </label>{{ book.id }}</div>
  <div>
    <label>Name: </label> <input [(ngModel)]="book.name"
placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

book-detail.component.css:

```
label {
  display: inline-block;
  width: 3em;
  margin: 0.5em 0;
  color: orange;
  font-weight: bold;
}
input {
  height: 2em;
  font-size: 1em;
  padding-left: 0.4em;
```

```
}
button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #ccc;
  cursor: auto;
}
h2{
  color:Orange;
}
```

page-not-found.component.html:

```
<div>
    <h1>404 Error</h1>
    <h1>Page Not Found</h1>
</div>
```

app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
const appRoutes: Routes = [
    { path: 'dashboard', component: DashboardComponent },
    { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
    { path: 'books', component: BookComponent },
    { path: 'detail/:id', component: BookDetailComponent },
    { path: '**', component: PageNotFoundComponent },
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
```

```
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

app.module.ts :

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { AppRoutingModule } from './app-routing.module';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule, FormsModule,
AppRoutingModule],
  declarations: [AppComponent, BookComponent,
DashboardComponent, BookDetailComponent,
PageNotFoundComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.component.ts:

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'Tour of Books';
}
```

app.component.html:

```
<h1>{{title}}</h1>
<nav>
    <a [routerLink]='["/dashboard"]'
routerLinkActive="active">Dashboard</a>
    <a [routerLink]='["/books"]'
routerLinkActive="active">Books</a>
```

```
</nav>
<router-outlet></router-outlet>
```

app.component.css:

```
/* Master Styles */
h1 {
  color: green;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
a {
  cursor: pointer;
  cursor: hand;
}
button {
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #aaa;
  cursor: auto;
}
/* Navigation link styles */
nav a {
  padding: 5px 10px;
  text-decoration: none;
  margin-right: 10px;
  margin-top: 10px;
```

```
  display: inline-block;
  background-color: #eee;
  border-radius: 4px;
}
nav a:visited, a:link {
  color: #607D8B;
}
nav a:hover {
  color: #039be5;
  background-color: #CFD8DC;
}
nav a.active {
  color: green;
}
/* everywhere else */
* {
  font-family: Arial, Helvetica, sans-serif;
}
```

styles.css :

```
body{
    padding:10px;
}
```

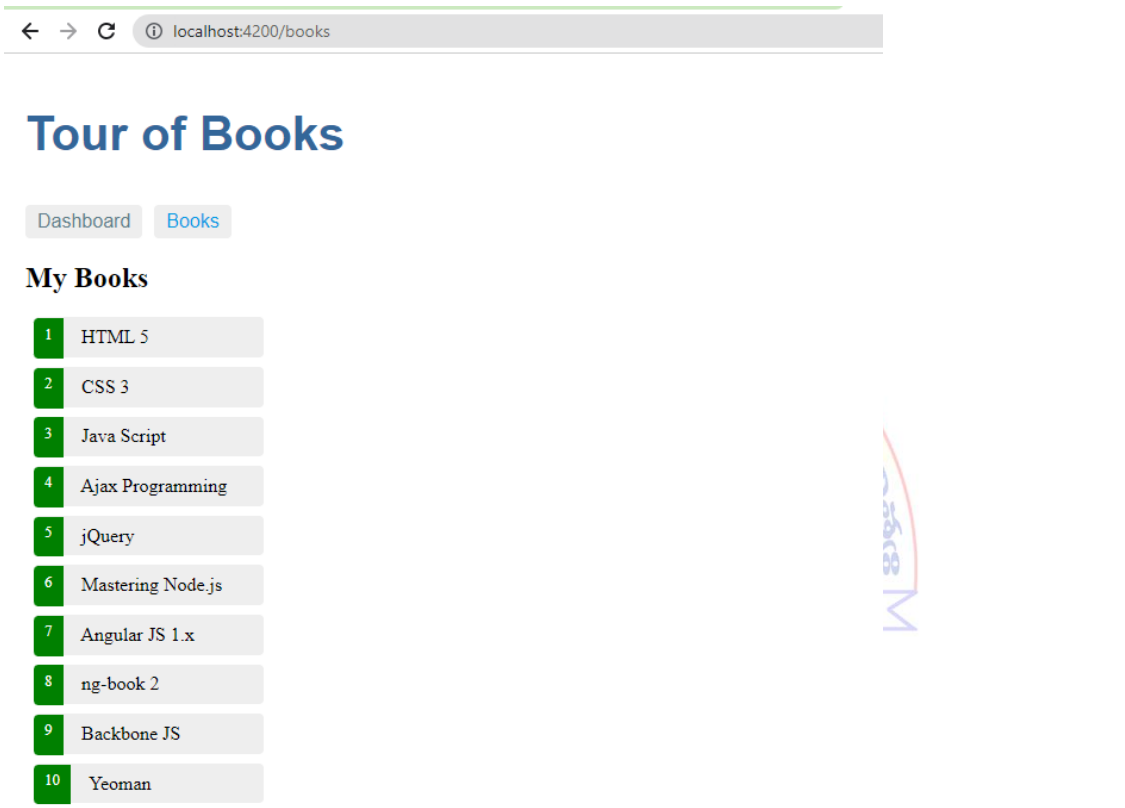
book.component.ts:

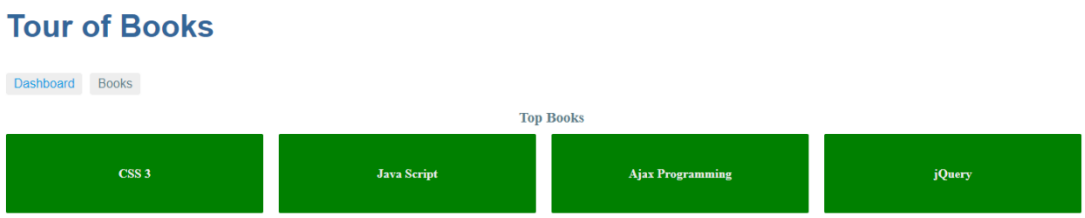
```
import { Component, OnInit } from '@angular/core';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  books!: Book[];
  errorMessage!: string;
  constructor(private bookService: BookService) { }
  getBooks() {
    this.bookService.getBooks().subscribe({
       next:  books => this.books = books,
       error:error => this.errorMessage = <any>error
     })
   }
  ngOnInit(): void {
    this.getBooks();
  }
}
```

book.component.html:

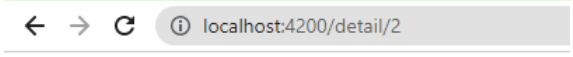
```
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{ book.id }}</span> {{book.name }}
  </li>
</ul>
<div class="error" *ngIf="errorMessage">{{ errorMessage
}}</div>
```

Output:





When clicked on CSS3 this page will Display.



Tour of Books

Dashboard Books

CSS 3 details!

Id: 2

Name: CSS 3

Back

10b) Course Name: Angular JS
Module Name: Route Guards
Considering the same example used for routing, add route guard to BooksComponent. Only after logging in, the user should be able to access BooksComponent. If the user tries to give the URL of Bookscomponent in another tab or window, or if the user tries.
Aim: To Implement Route Guards.
Description:
Angular route guards are interfaces provided by Angular which, when implemented, allow us to control the accessibility of a route based on conditions provided in class implementation of that interface.
Here are some types of Angular guards: CanActivate, CanActivateChild, CanLoad, CanDeactivate and Resolve.
Using canActivate, access can be permitted to only authenticated users.
Syntax:
Class GuardService implements CanActivate{
 canActivate( ): boolean {
 }
}

Program:
login.component.html:

```
<h3 style="position: relative; left: 60px; color:green">Login
Form</h3>
<div *ngIf="invalidCredentialMsg" style="color: red">
  {{ invalidCredentialMsg }}
</div>
<br />
<div style="position: relative; left: 20px">
  <form [formGroup]="loginForm" (ngSubmit)="onFormSubmit()">
    <p>User Name <input formControlName="username" /></p>
    <p>
      Password
      <input
        type="password"
        formControlName="password"
        style="position: relative; left: 10px"
      />
    </p>
    <p><button class="btn-success"
type="submit">Submit</button></p>
  </form>
</div>
```

login.component.ts:

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { Router } from '@angular/router';
import { LoginService } from './login.service';
```

```
@Component({
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'],
})
export class LoginComponent {
  invalidCredentialMsg!: string;
  loginForm!: FormGroup;
  constructor(
    private loginService: LoginService,
    private router: Router,
    private formbuilder: FormBuilder
  ) {
    this.loginForm = this.formbuilder.group({
      username: [],
      password: [],
    });
  }
  onFormSubmit(): void {
    const uname = this.loginForm.value.username;
    const pwd = this.loginForm.value.password;
    this.loginService
      .isUserAuthenticated(uname, pwd)
      .subscribe({next:(authenticated) => {
        if (authenticated) {
          this.router.navigate(['/books']);
        } else {
          this.invalidCredentialMsg = 'Invalid Credentials.
Try again.';
        }
      }});
  }
}
```

user.ts:

```
export class User {
    constructor(public userId: number, public username:
string, public password: string) { }
  }
```

login.service.ts:

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';
import { User } from './user';
const USERS = [
    new User(1, 'user1', 'pass1'),
    new User(2, 'user2', 'pass2')
];
const usersObservable = of(USERS);
@Injectable({
    providedIn: 'root'
})
```

```
export class LoginService {
    private isloggedIn = false;
    getAllUsers(): Observable<User[]> {
        return usersObservable;
    }
    isUserAuthenticated(username: string, password: string):
Observable<boolean> {
        return this.getAllUsers().pipe(
            map(users => {
                const Authenticateduser = users.find(user =>
(user.username === username) && (user.password === password));
                if (Authenticateduser) {
                    this.isloggedIn = true;
                } else {
                    this.isloggedIn = false;
                }
                return this.isloggedIn;
            })
        );
    }
    isUserLoggedIn(): boolean {
        return this.isloggedIn;
    }
}
```

login-guard.service.ts:

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';
import { LoginService } from './login.service';
@Injectable({
    providedIn: 'root'
})
export class LoginGuardService implements CanActivate {
    constructor(private loginService: LoginService, private
router: Router) { }
    canActivate(): boolean {
        if (this.loginService.isUserLoggedIn()) {
            return true;
        }
        this.router.navigate(['/login']);
        return false;
    }
}
```

app.module.ts:

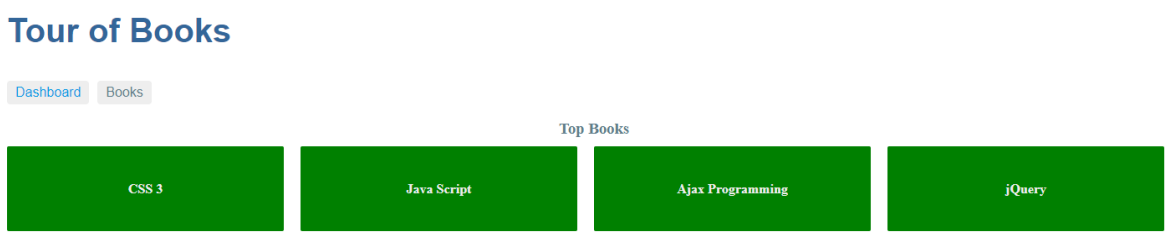
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from
'@angular/forms';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
```

```
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { AppRoutingModule } from './app-routing.module';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
import { LoginComponent } from './login/login.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule,
ReactiveFormsModule,FormsModule, AppRoutingModule],
  declarations: [AppComponent, LoginComponent, BookComponent,
DashboardComponent, BookDetailComponent,
PageNotFoundComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

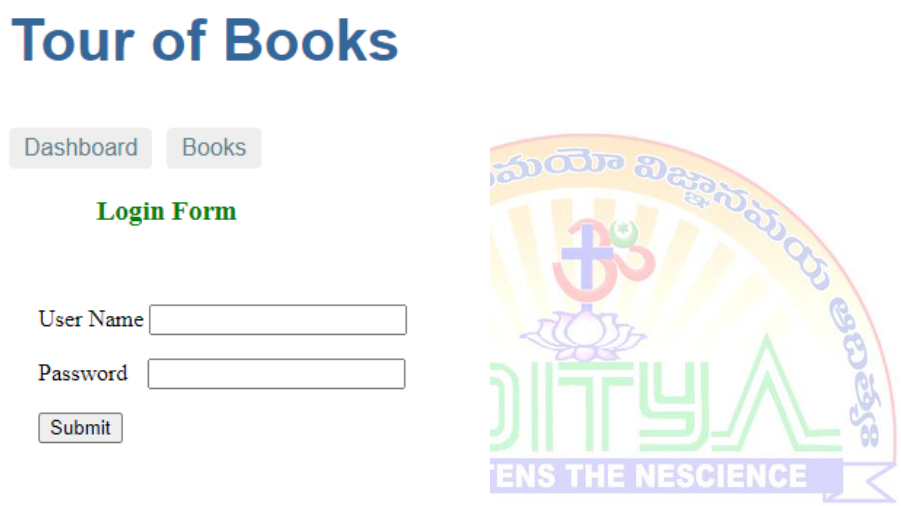
app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
import { LoginGuardService } from './login/login-
guard.service';
import { LoginComponent } from './login/login.component';
const appRoutes: Routes = [
    { path: 'dashboard', component: DashboardComponent },
    { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
    { path: 'books', component: BookComponent ,
canActivate:[LoginGuardService] },
    { path: 'detail/:id', component: BookDetailComponent },
    {path: 'login',component:LoginComponent},
    { path: '**', component: PageNotFoundComponent },
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

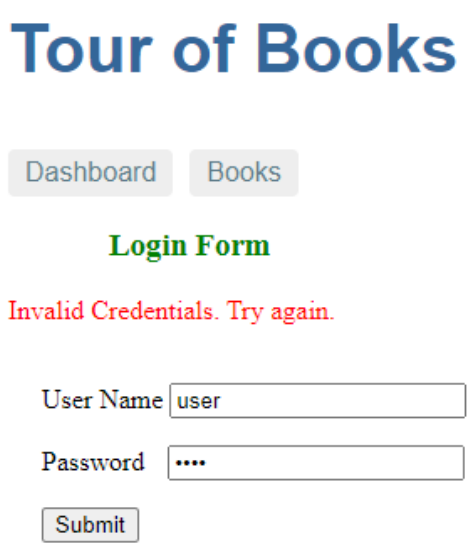
Output:



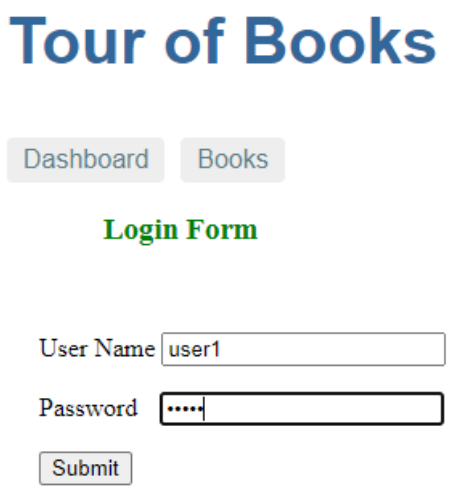
When Clicked on Books it displays login form as follows:



When invalid credentials are provided it displays like this:



Filling the form with Correct Credentials:



When login form submitted then books will display like this:

localhost:4200/books

Tour of Books

Dashboard Books

My Books

1. HTML 5
2. CSS 3
3. Java Script
4. Ajax Programming
5. jQuery
6. Mastering Node.js
7. Angular JS 1.x
8. ng-book 2
9. Backbone JS
10. Yeoman

10c) Course Name: Angular JS
Module Name: Asynchronous Routing
Apply lazy loading to BookComponent. If lazy loading is not added to the demo, it has loaded in 1.14 s. Observe the load time at the bottom of the browser console. Press F12 in the browser and click the Network tab and check the Load time.
Aim: To Implement Asynchronous Routing.

Description:
Asynchronous routing is the most straightforward method of routing signals. Any asynchronous route can be defined in terms of two signals: a source and a destination.

When an Angular application has a lot of components, it will increase the size of the application. In such cases, the application takes a lot of time to load.
To overcome this problem, asynchronous routing is preferred, i.e, modules must be loaded lazily only when they are required instead of loading them at the beginning of the execution

Lazy Loading has the following benefits:
- Modules are loaded only when the user requests for it
- Load time can be speeded up for users who will be visiting only certain areas of the application.

Lazy Loading Route Configuration:
To apply lazy loading on modules, create a separate routing configuration file for that module and map an empty path to the component of that module.

Program:

book-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book.component';
import { LoginGuardService } from '../login/login-
guard.service';
const bookRoutes: Routes = [
    {
        path: '',
        component: BookComponent,
        canActivate: [LoginGuardService]
    }
];
@NgModule({
    imports: [RouterModule.forChild(bookRoutes)],
    exports: [RouterModule]
})
export class BookRoutingModule { }
```

book.module.ts:

```
import { NgModule } from '@angular/core';
```

```
import { CommonModule } from '@angular/common';
import { BookComponent } from './book.component';
import { BookRoutingModule } from './book-routing.module';
@NgModule({
  imports: [CommonModule, BookRoutingModule],
  declarations: [BookComponent]
})
export class BookModule { }
```

app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
import { LoginGuardService } from './login/login-
guard.service';
import { LoginComponent } from './login/login.component';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
const appRoutes: Routes = [
    { path: '', redirectTo: '/login', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'books', loadChildren: () =>
import('./book/book.module').then(m => m.BookModule) },
    { path: 'dashboard', component: DashboardComponent },
    { path: 'detail/:id', component: BookDetailComponent} ,
    { path: '**', component: PageNotFoundComponent }
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

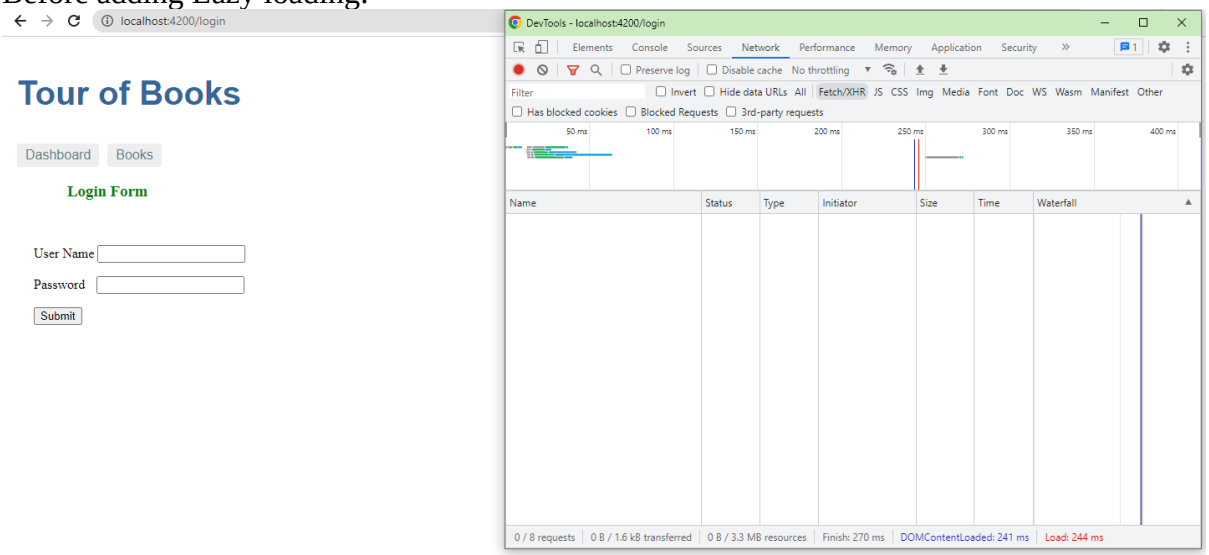
app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from
'@angular/forms';
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { DashboardComponent } from
'./dashboard/dashboard.component';
```

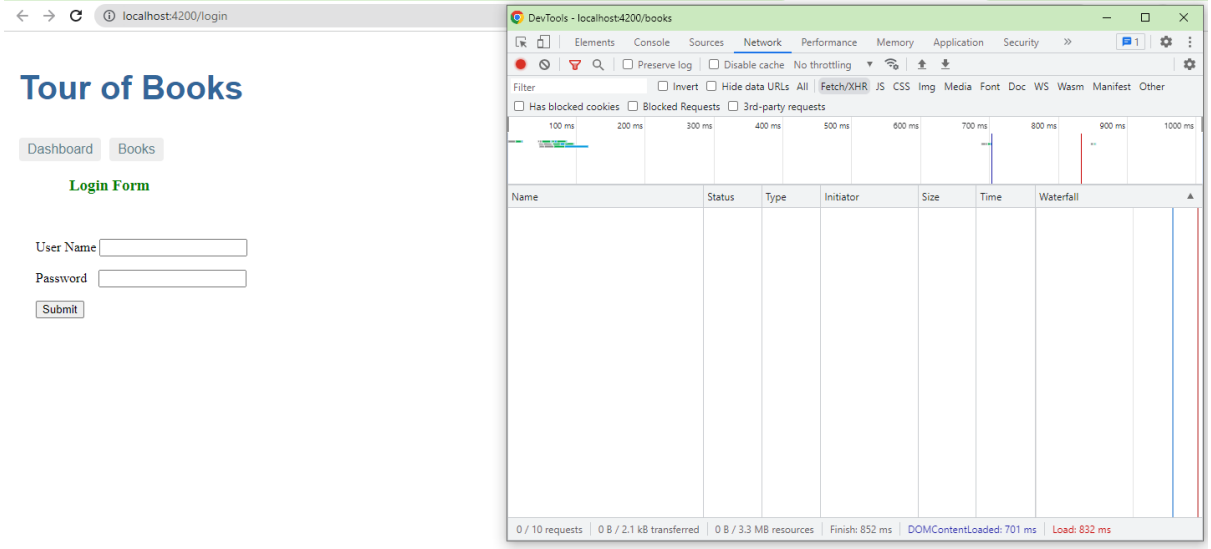
```
import { BookDetailComponent } from './book-detail/book-
detail.component';
import { AppRoutingModule } from './app-routing.module';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
import { LoginComponent } from './login/login.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule,
ReactiveFormsModule,FormsModule, AppRoutingModule],
  declarations: [AppComponent, LoginComponent,
DashboardComponent, BookDetailComponent,
PageNotFoundComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Output:

Before adding Lazy loading:



After adding Lazy loading:


localhost:4200/login
Tour of Books
Dashboard
Books
Login Form
User Name
Password
Submit
DevTools - localhost:4200/books
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All
Fetch/XHR
JS
CSS
Img
Media
Font
Doc
WS
Wasm
Manifest
Other
Has blocked cookies
Blocked Requests
3rd-party requests
Name
Status
Type
Initiator
Size
Time
Waterfall
0 / 10 requests
0 B / 2.1 kB transferred
0 B / 3.3 MB resources
Finish: 852 ms
DOMContentLoaded: 701 ms
Load: 832 ms

10d) Course Name: Angular JS
Module Name: Nested Routes
Implement Child Routes to a submodule.

Aim: To Implement Child Routes to a submodule.

Description:

Nested Routes:

In Angular, you can also create sub-routes or child routes for your components which means in an application there will be one root route just like a root component/root module and other routes will be configured for their respective components. Configuring routes module-wise is the best practice to make modular Angular applications.

Syntax:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { ParentComponent } from './parent.component';
import { ChildComponent } from './child.component';
const routes: Routes = [
  {
    path: 'parent',
    component: ParentComponent,
    children: [
      {
        path: 'child',
        component: ChildComponent
      }
    ]
  }
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class ParentRoutingModule { }
```

Program:

app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { LoginComponent } from './login/login.component';
@NgModule({
  imports: [BrowserModule, HttpClientModule,
ReactiveFormsModule, AppRoutingModule],
  declarations: [AppComponent, LoginComponent],
  providers: [],
  bootstrap: [AppComponent]
```

```
})
export class AppModule { }
```

app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { PageNotFoundComponent } from './page-not-found/page-
not-found.component';
const appRoutes: Routes = [
    { path: '', redirectTo: '/login', pathMatch: 'full' },
    { path: 'login', component: LoginComponent },
    { path: 'books', loadChildren: () =>
import('./book/book.module').then(m => m.BookModule) },
    { path: '**', component: PageNotFoundComponent }
];
@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

app.component.html:

```
<h1>{{title}}</h1>
<nav>
    <a [routerLink]='["/books"]'
routerLinkActive="active">Books</a>
    <a [routerLink]='["/books/dashboard"]'
routerLinkActive="active">Dashboard</a>
</nav>
<router-outlet></router-outlet>
```

book.module.ts:

```
import { NgModule } from '@angular/core';
import { BookComponent } from './book.component';
import { BookRoutingModule } from './book-routing.module';
import { FormsModule } from '@angular/forms';
import { BookDetailComponent } from '../book-detail/book-
detail.component';
import { DashboardComponent } from
'../dashboard/dashboard.component';
import { CommonModule } from '@angular/common';
@NgModule({
  imports: [ CommonModule, BookRoutingModule, FormsModule],
```

```
  declarations: [BookComponent, BookDetailComponent,
DashboardComponent]
})
export class BookModule { }
```

book-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { BookComponent } from './book.component';
import { LoginGuardService } from '../login/login-
guard.service';
import { DashboardComponent } from
'../dashboard/dashboard.component';
import { BookDetailComponent } from '../book-detail/book-
detail.component';
const bookRoutes: Routes = [
    {
      path: '',
      component: BookComponent,
      children: [
        { path: 'dashboard', component: DashboardComponent },
        { path: 'detail/:id', component: BookDetailComponent }
      ],
      canActivate: [LoginGuardService]
    }];
@NgModule({
    imports: [RouterModule.forChild(bookRoutes)],
    exports: [RouterModule]
})
export class BookRoutingModule { }
```

book.component.html:

```
<br/>
     <h2>MyBooks</h2>
     <ul class="books">
       <li *ngFor="let book of books "
(click)="gotoDetail(book)">
          <span class="badge">{{book.id}}</span> {{book.name}}
       </li>
     </ul>
     <div>
       <router-outlet></router-outlet>
     </div>
     <div class="error"
*ngIf="errorMessage">{{errorMessage}}</div>
```

book.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Book } from './book';
import { BookService } from './book.service';
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {
  books: Book[]=[];
  errorMessage!: string;
  constructor(private bookService: BookService, private
router: Router) { }
  getBooks() {
    this.bookService.getBooks().subscribe({
       next:  books => {console.log(books);this.books =
books},
       error:error => this.errorMessage = <any>error
     })
   }
   gotoDetail(book: Book): void {
    this.router.navigate(['/books/detail/', book.id]);
  }
  ngOnInit(): void {
    this.getBooks();
  }
}
```

book-detail.component.html:

```
<div *ngIf="book">
  <h2>{{ book.name }} details!</h2>
  <div><label>Id: </label>{{ book.id }}</div>
  <div>
    <label>Name: </label> <input [(ngModel)]="book.name"
placeholder="name" />
  </div>
  <button (click)="goBack()">Back</button>
</div>
```

book-detail.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-book-detail',
```

```
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
})
export class BookDetailComponent implements OnInit {
  book!: Book;
  error!: any;
  constructor(
    private bookService: BookService,
    private route: ActivatedRoute
  ) { }
  ngOnInit() {

      this.route.paramMap.subscribe(params => {

this.bookService.getBook(params.get('id')).subscribe((book) =>
{
       this.book = book ?? this.book;
      });
    });
  }
  goBack() {
    window.history.back();
  }
}
```

dashboard.component.html:

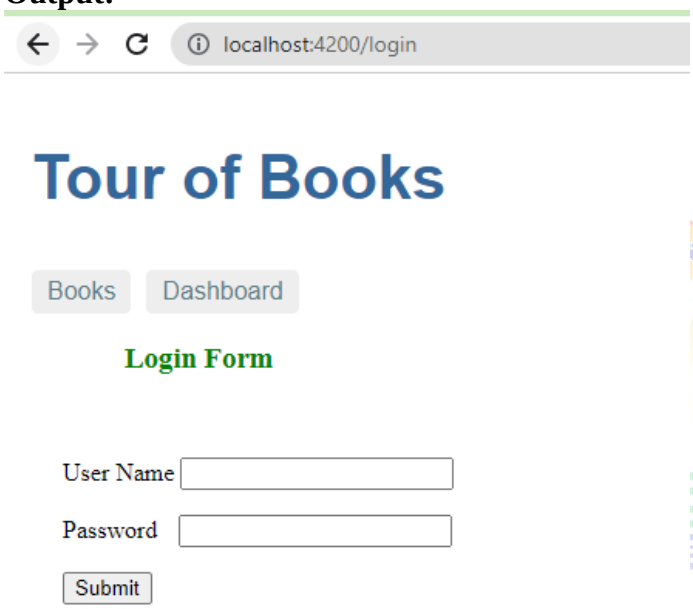
```
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)"
class="col-1-4">
    <div class="module book">
      <h4>{{ book.name }}</h4>
    </div>
  </div>
</div>
```

dashboard.component.ts:

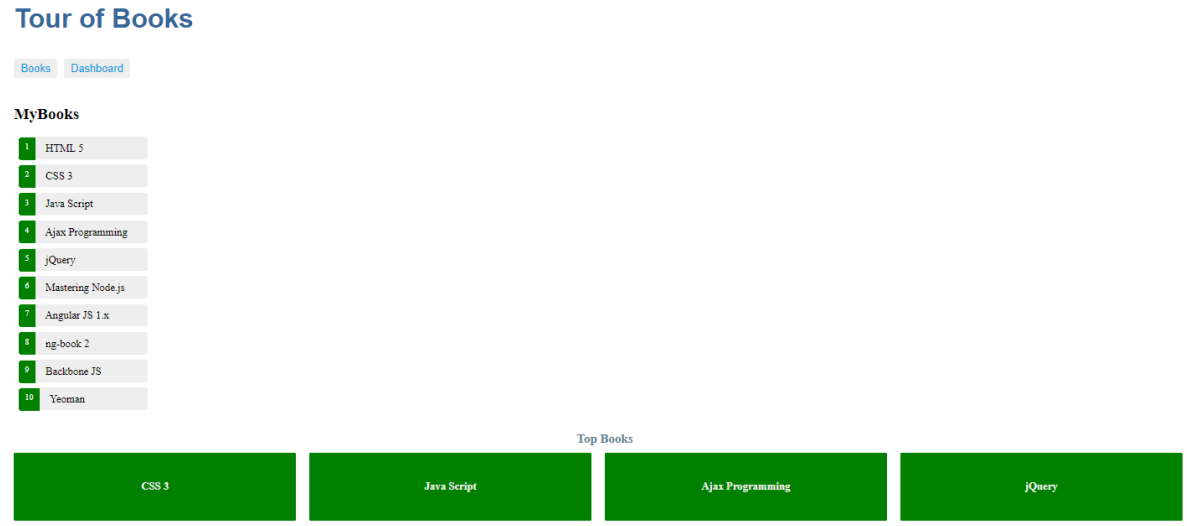
```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Book } from '../book/book';
import { BookService } from '../book/book.service';
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
```

```
  books: Book[] = [];
  constructor(
    private router: Router,
    private bookService: BookService) { }
  ngOnInit(): void {
    this.bookService.getBooks()
      .subscribe(books => this.books = books.slice(1, 5));
  }
  gotoDetail(book: Book): void {
    this.router.navigate(['/books/detail', book.id]);
  }
}
```

Output:




localhost:4200/books/detail/1
Tour of Books
Books
Dashboard
MyBooks
1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman
HTML 5 details!
Id: 1
Name: HTML 5
Back


Tour of Books
Books
Dashboard
MyBooks
1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman
Top Books
CSS 3
Java Script
Ajax Programming
jQuery


Tour of Books
Books
Dashboard
MyBooks
1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman
Java Script details!
Id: 3
Name: Java Script
Back


localhost:4200/books
Tour of Books
Books
Dashboard
MyBooks
1 HTML 5
2 CSS 3
3 Java Script
4 Ajax Programming
5 jQuery
6 Mastering Node.js
7 Angular JS 1.x
8 ng-book 2
9 Backbone JS
10 Yeoman

11.a Course Name: MongoDB Essentials - A Complete MongoDB Guide Module Name: Installing MongoDB on the local computer, Create MongoDB Atlas Cluster Install MongoDB and configure ATLAS.

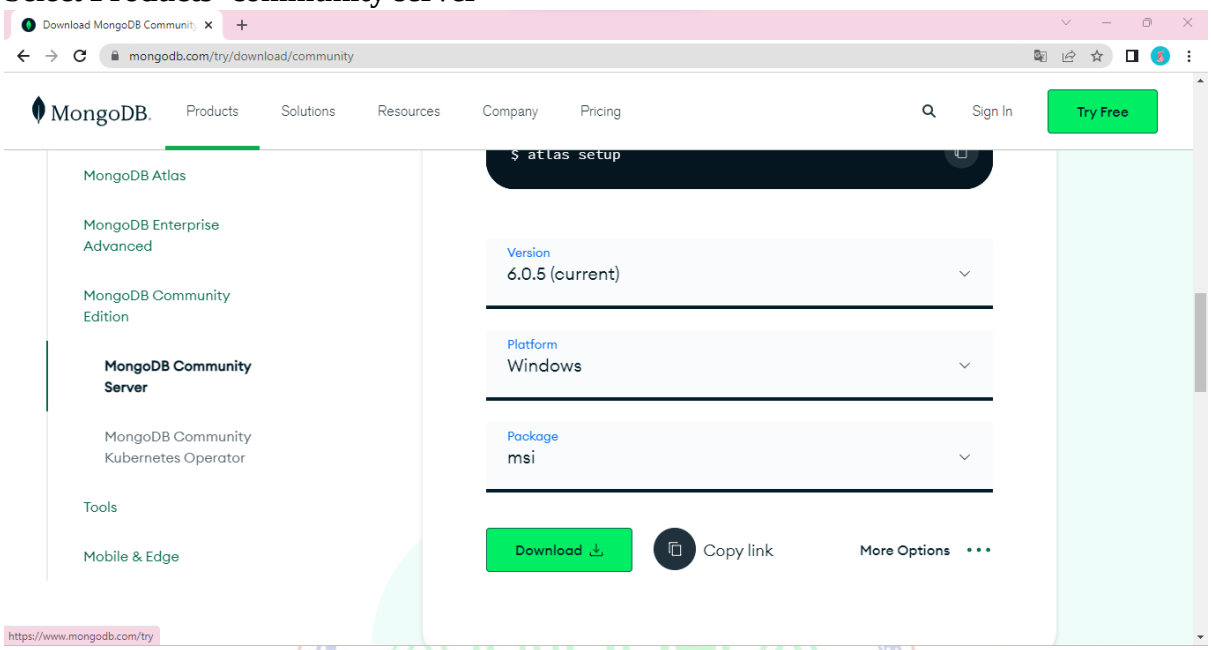
Aim: To Install MongoDB on the local computer, Create MongoDB Atlas Cluster Install MongoDB and configure ATLAS.

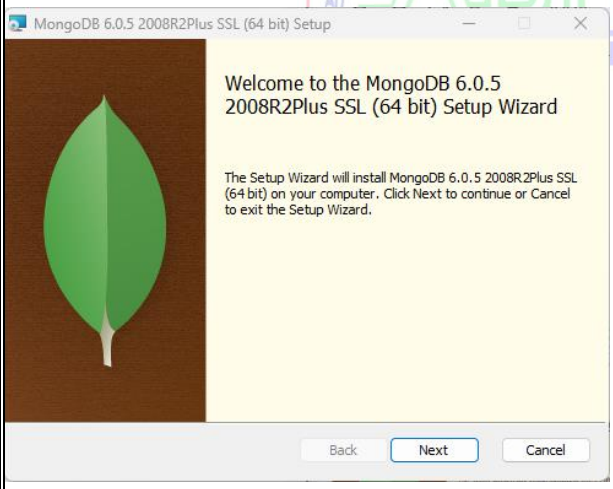
Process:

INSTALLING MONGODB locally:

Go to **mongodb.com**

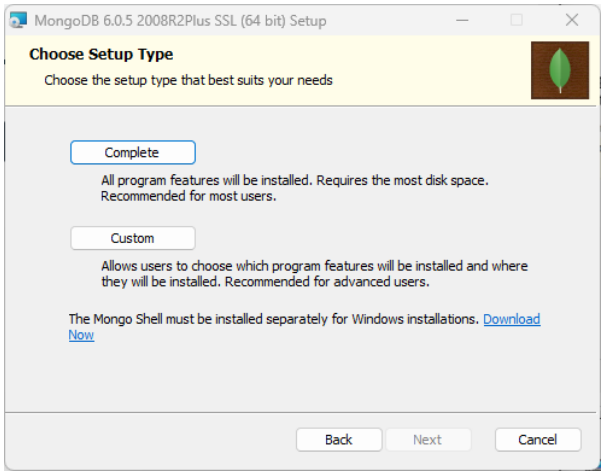
Select **Products>community server**



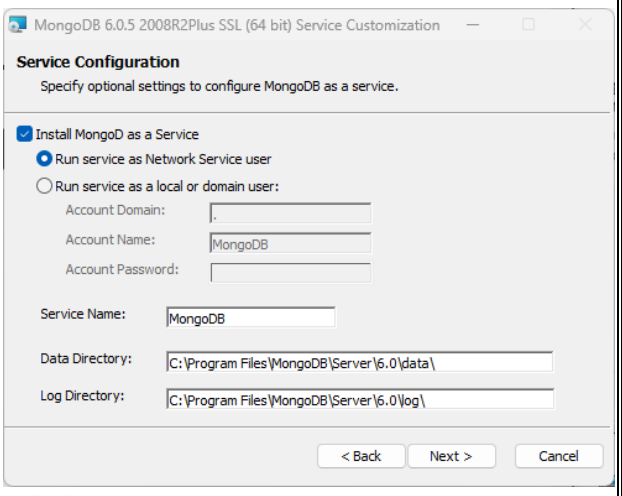


Click on Next

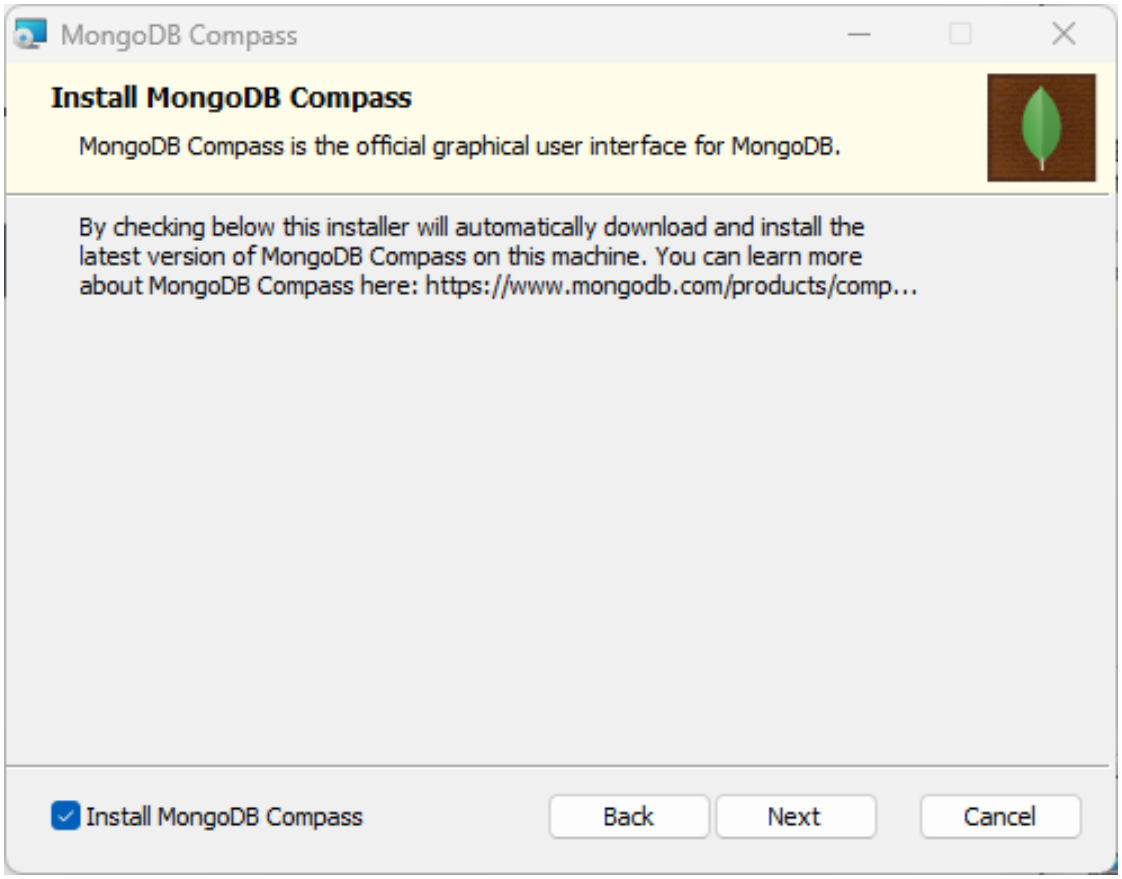
Click on Next



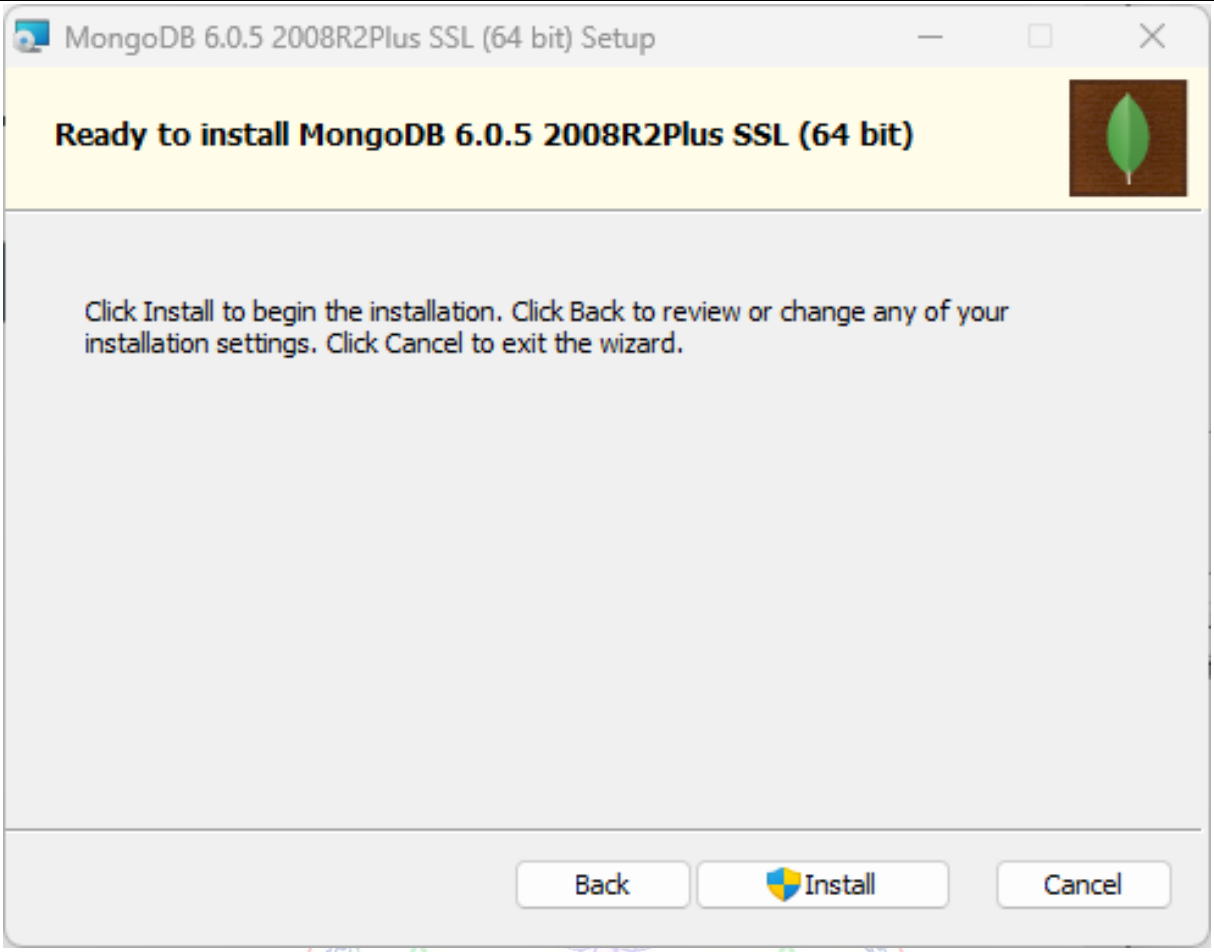
Click on complete

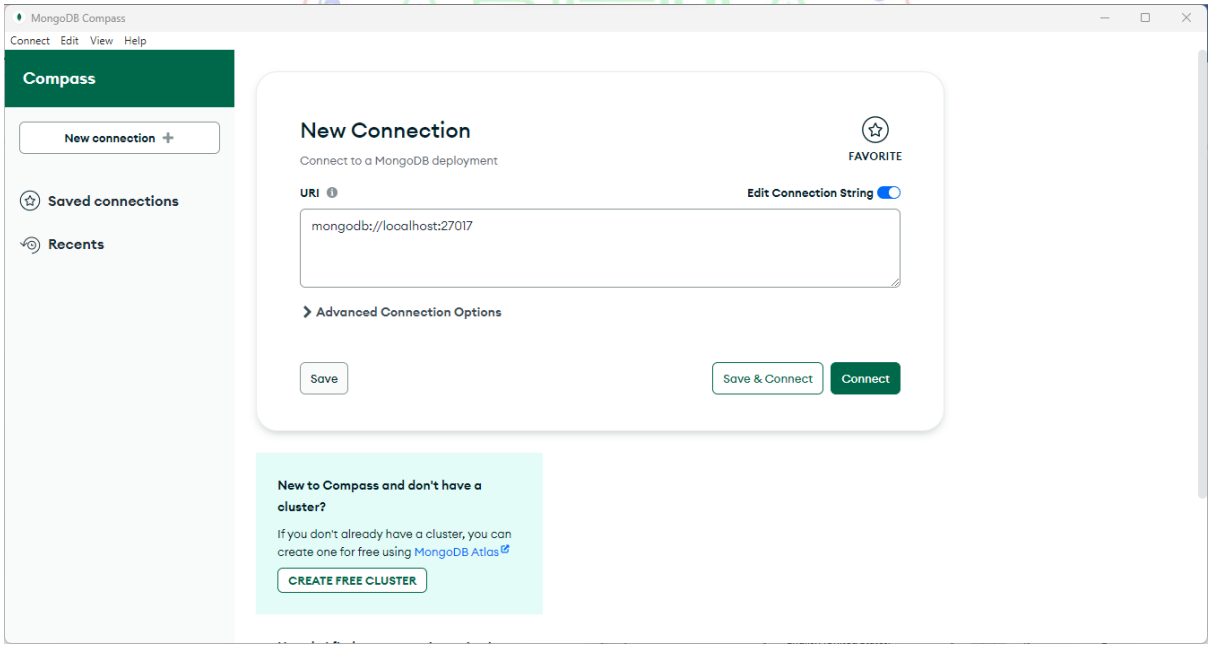


Click on next



Click on next and click install in the next step
After Installation set the path in the environment variables.

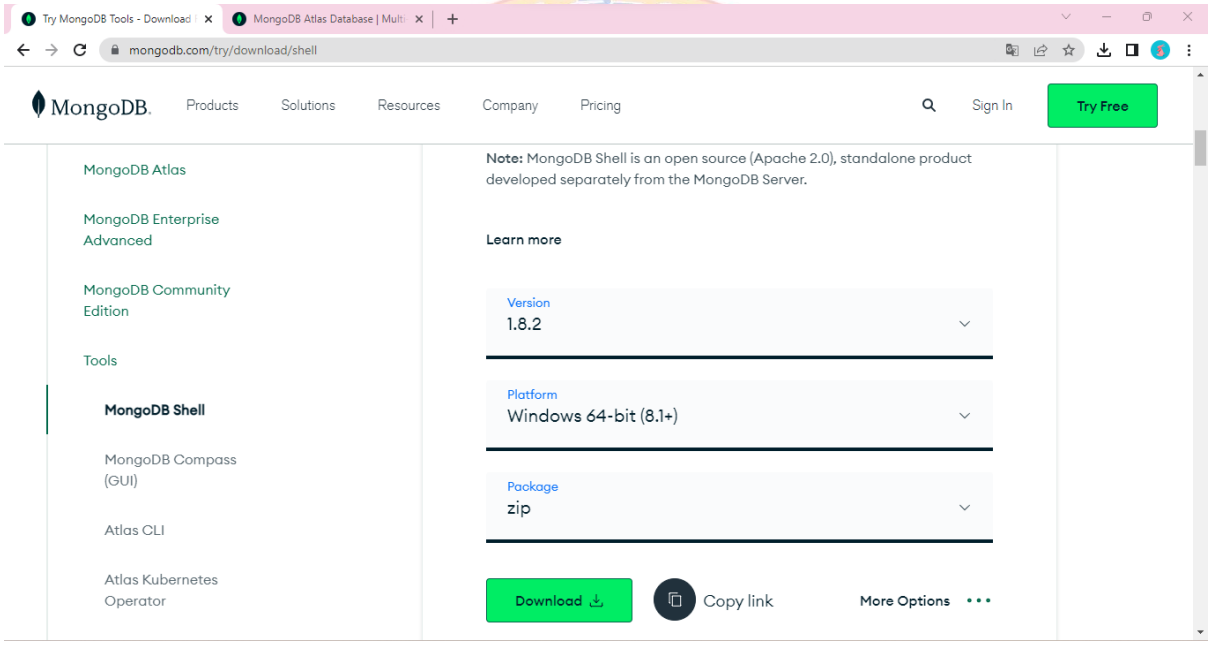

MongoDB 6.0.5 2008R2Plus SSL (64 bit) Setup
Ready to install MongoDB 6.0.5 2008R2Plus SSL (64 bit)
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.
Back
Install
Cancel


MongoDB Compass
Connect Edit View Help
Compass
New connection
Saved connections
Recents
New Connection
Connect to a MongoDB deployment
FAVORITE
URI
Edit Connection String
mongodb://localhost:27017
Advanced Connection Options
Save
Save & Connect
Connect
New to Compass and don't have a cluster?
If you don't already have a cluster, you can create one for free using MongoDB Atlas
CREATE FREE CLUSTER

Go to command prompt:

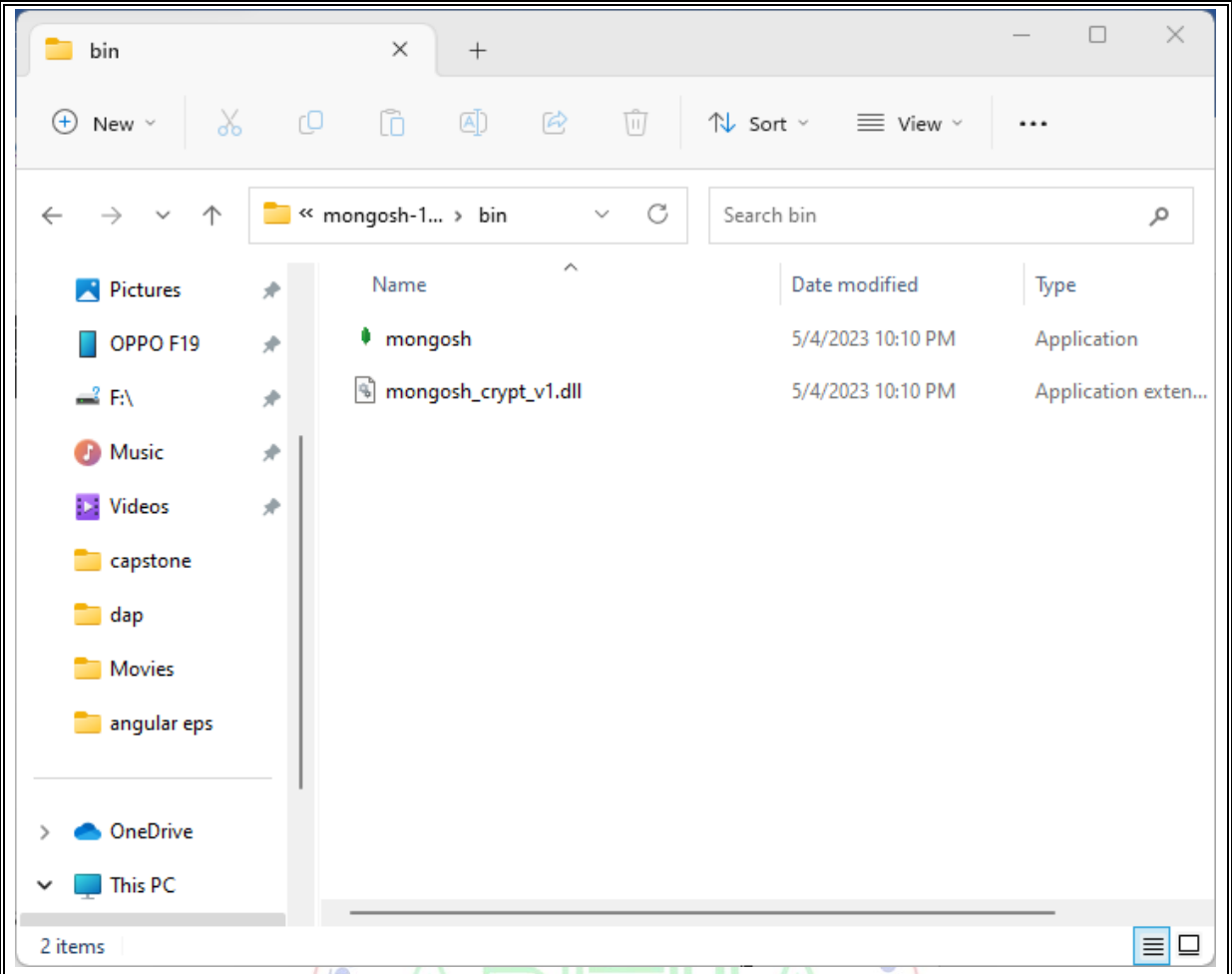
```
C:\Users\durga>mongod --version
db version v6.0.5
Build Info: {
    "version": "6.0.5",
    "gitVersion": "c9a99c120371d4d4c52cbb15dac34a36ce8d3b1d",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}

C:\Users\durga>mongod
{"t":{"$date":"2023-05-04T22:01:35.351+05:30"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"-","msg":"Automatically di
sabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2023-05-04T22:01:35.354+05:30"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"thread1","msg":"Initialize
d wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInterna
lClient":{"minWireVersion":0,"maxWireVersion":17},"outgoing":{"minWireVersion":6,"maxWireVersion":17},"isInternalClient"
:true}}}
{"t":{"$date":"2023-05-04T22:01:35.355+05:30"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx":"thread1","msg":"Implicit T
CP FastOpen in use."}
{"t":{"$date":"2023-05-04T22:01:35.356+05:30"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","namespace":"config.tenantMigrationDon
ors"}}
{"t":{"$date":"2023-05-04T22:01:35.356+05:30"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","namespace":"config.tenantMigratio
nRecipients"}}
```

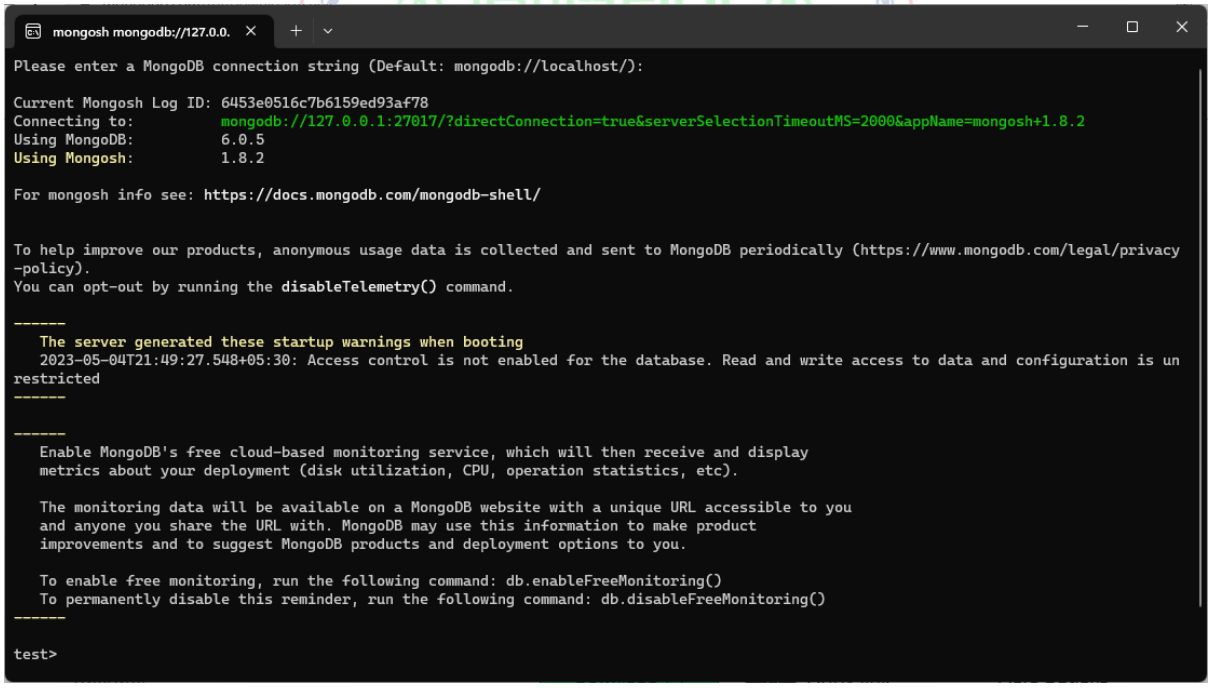
To download the shell:



Then extract the zip folder to **C:\mongosh** and set the path in the environmental variables.

Go to the **C:\mongosh\mongosh-1.8.0-win32-x64\bin** and click on mongosh:

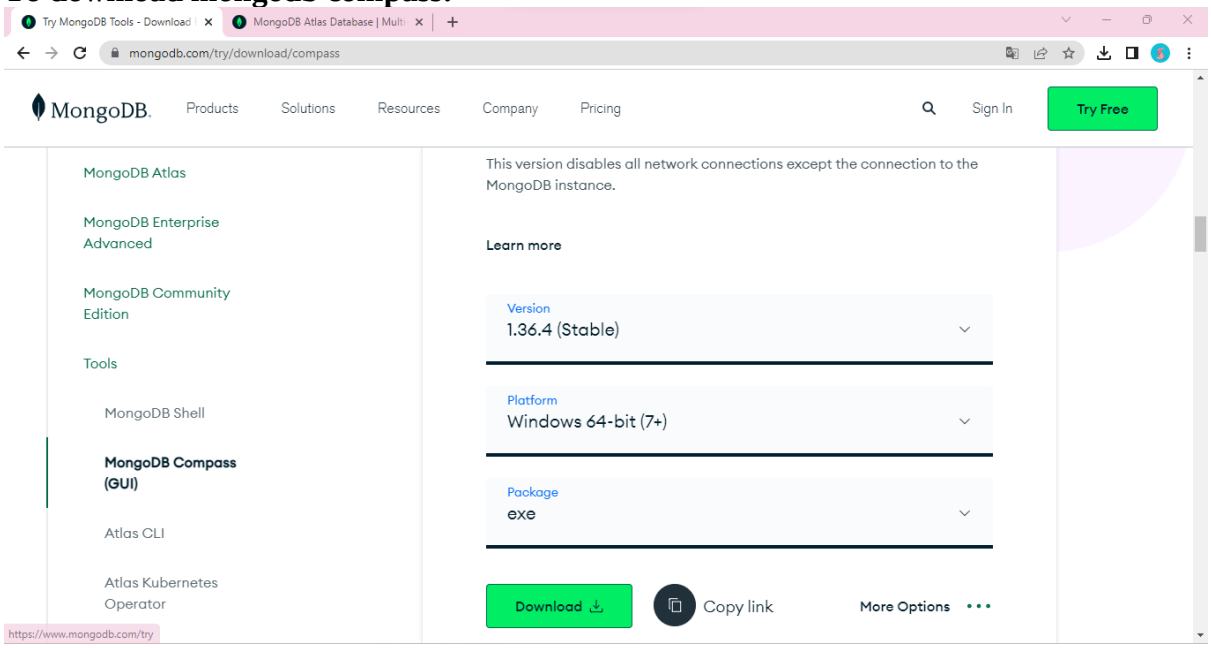

bin
New
Sort
View
« mongosh-1... › bin
Search bin
Pictures
OPPO F19
F:\
Music
Videos
capstone
dap
Movies
angular eps
OneDrive
This PC
Name
Date modified
Type
mongosh
5/4/2023 10:10 PM
Application
mongosh_crypt_v1.dll
5/4/2023 10:10 PM
Application exten...
2 items


mongosh mongodb://127.0.0.
Please enter a MongoDB connection string (Default: mongodb://localhost/):
Current Mongosh Log ID: 6453e0516c7b6159ed93af78
Connecting to: mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.2
Using MongoDB: 6.0.5
Using Mongosh: 1.8.2
For mongosh info see: https://docs.mongodb.com/mongodb-shell/
To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

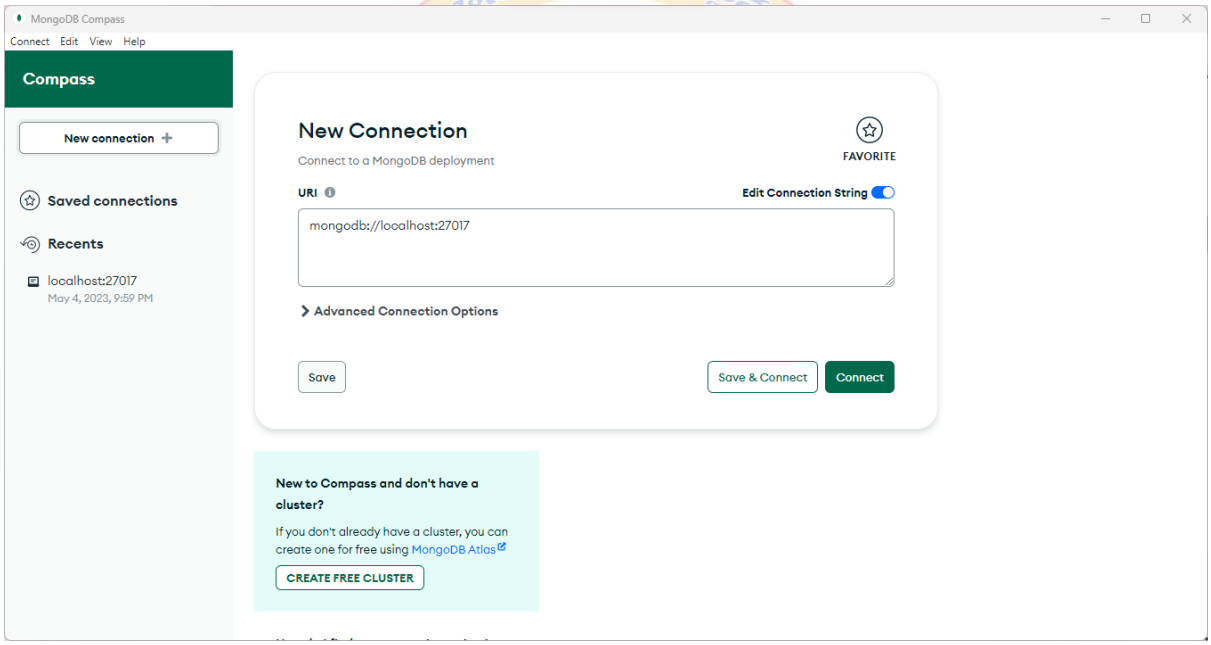
The server generated these startup warnings when booting
2023-05-04T21:49:27.548+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---

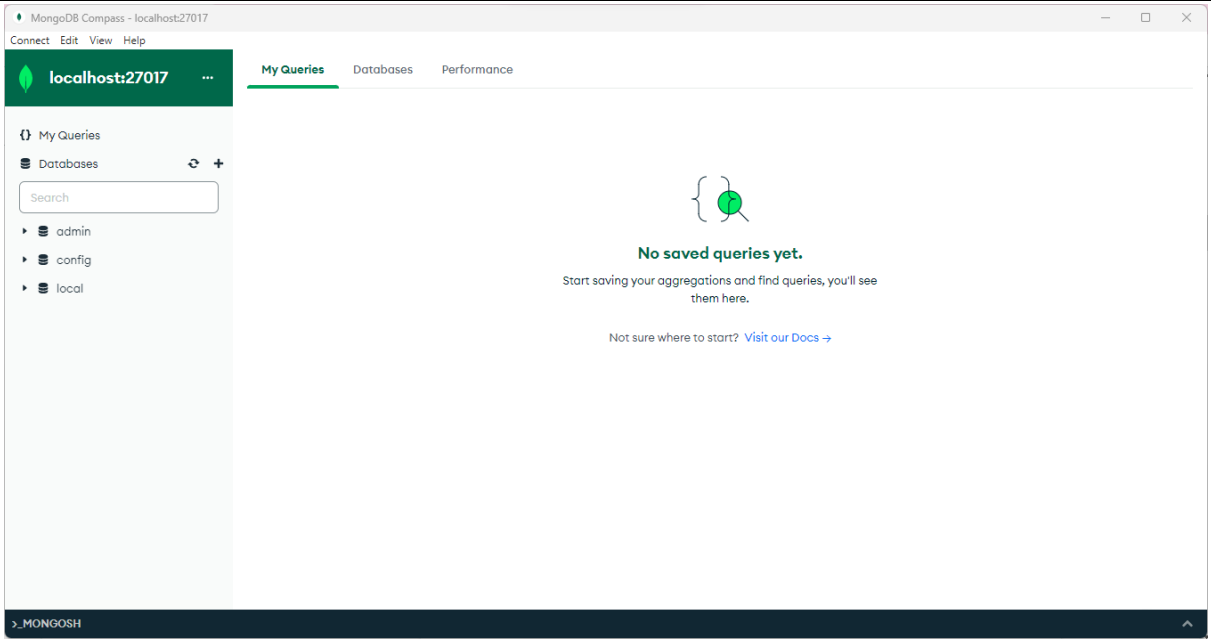
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
test>

To download mongodb compass:



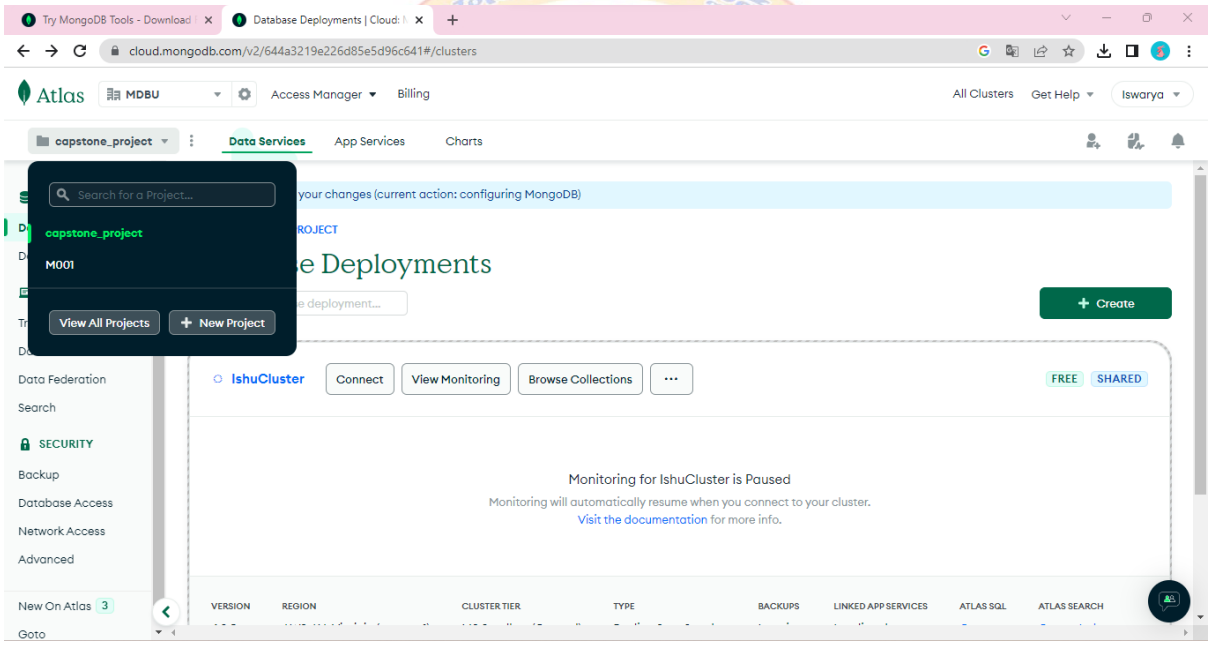
Install the mongodb compass



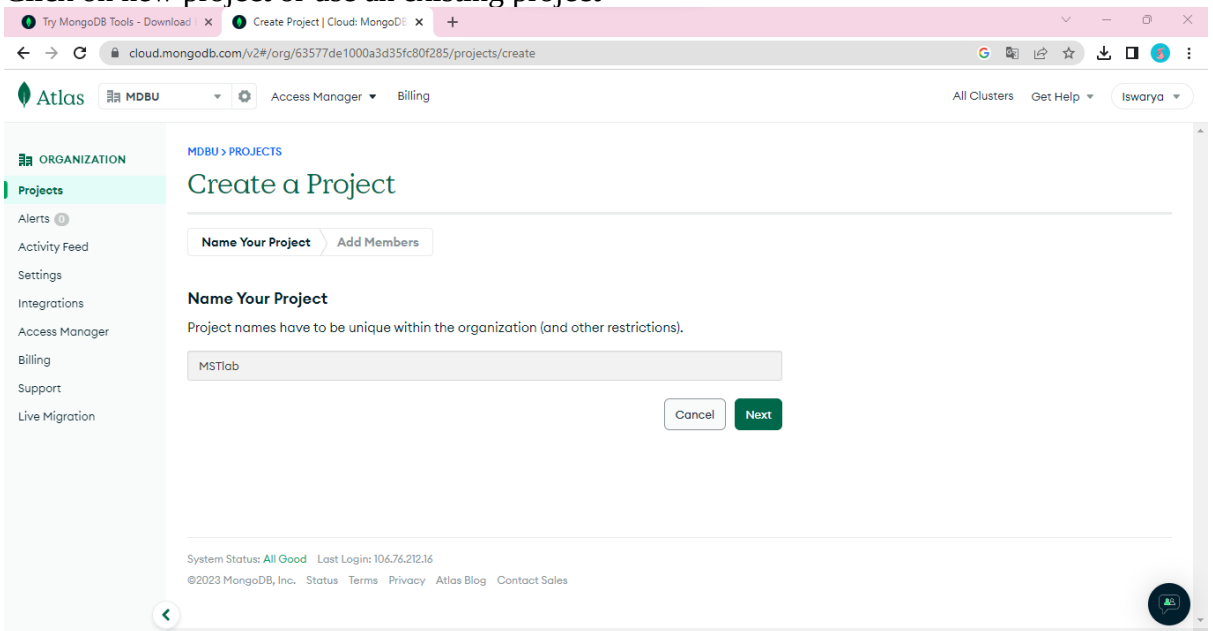


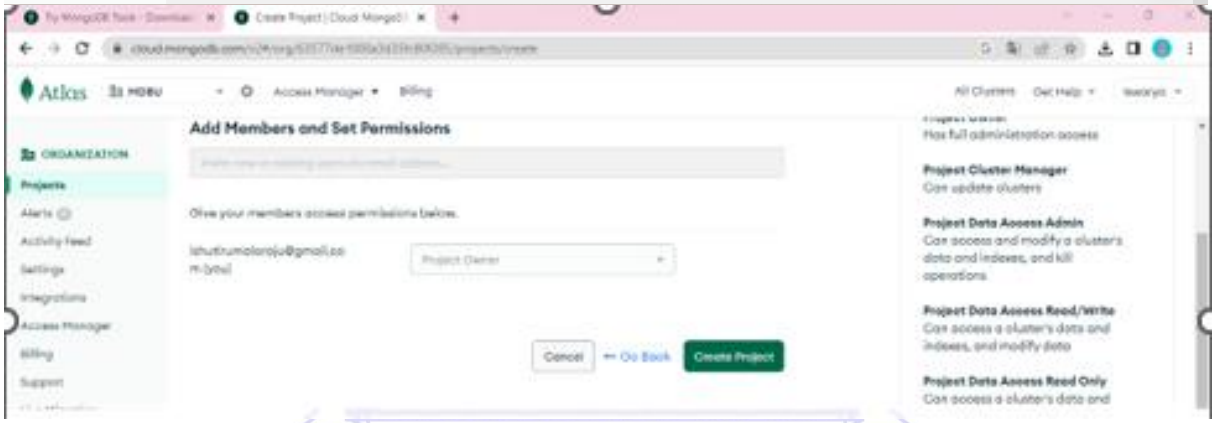
CREATING CLUSTER IN ATLAS:

Go to mongodb Atlas and signup or login

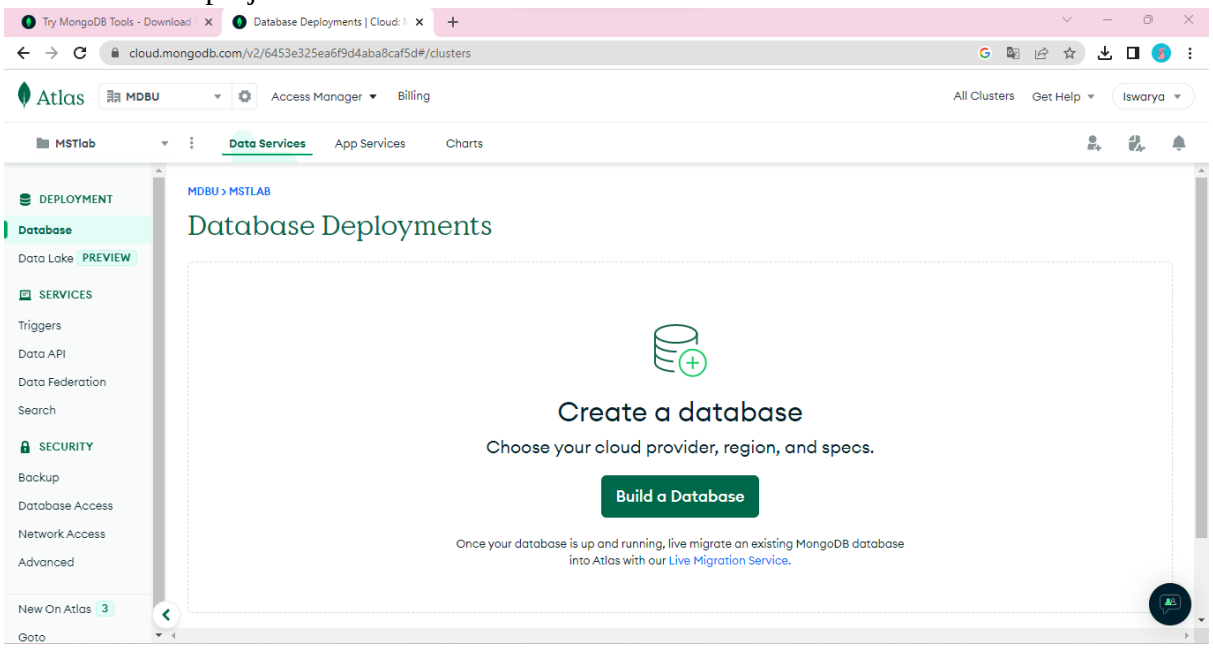


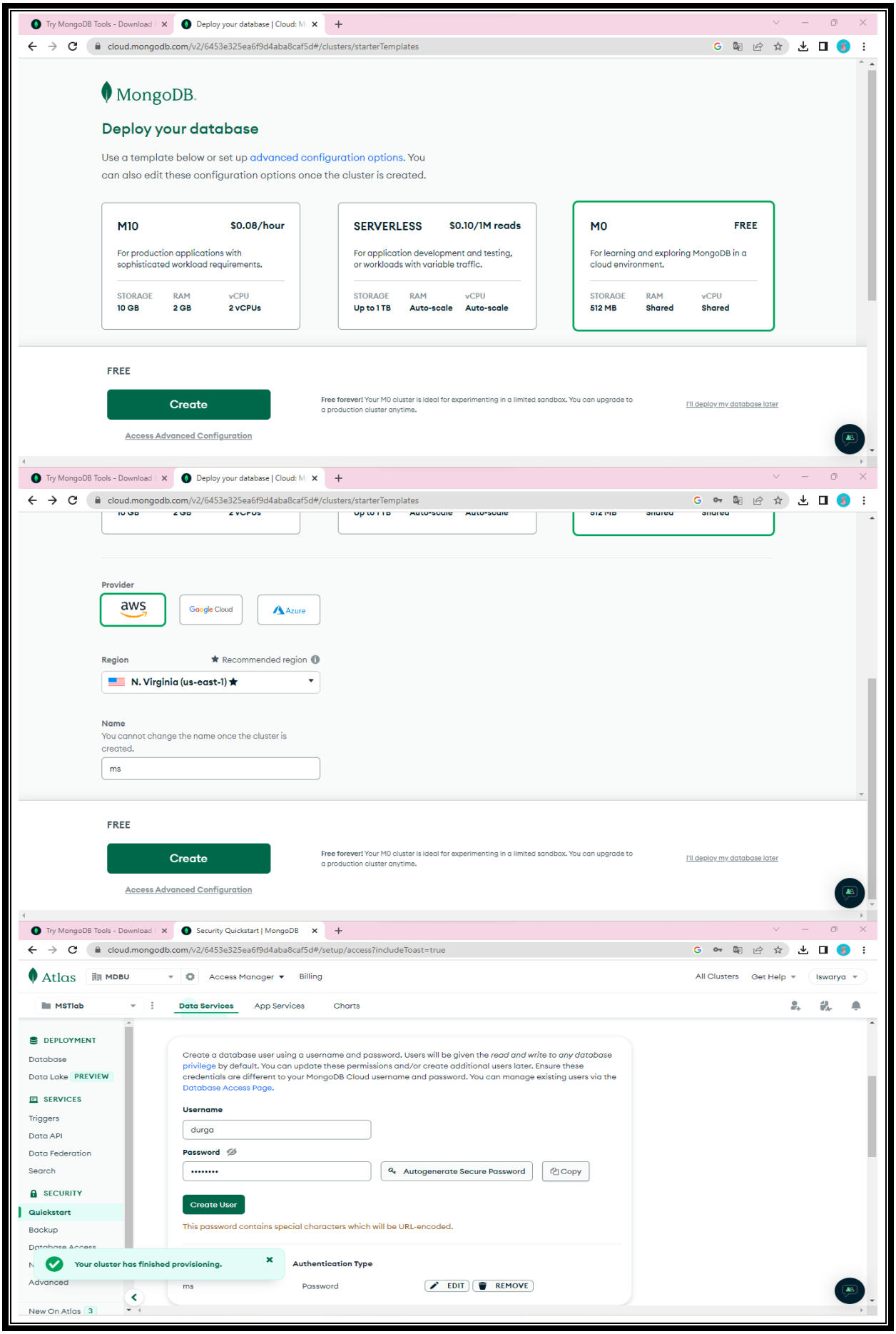
Click on new project or use an existing project

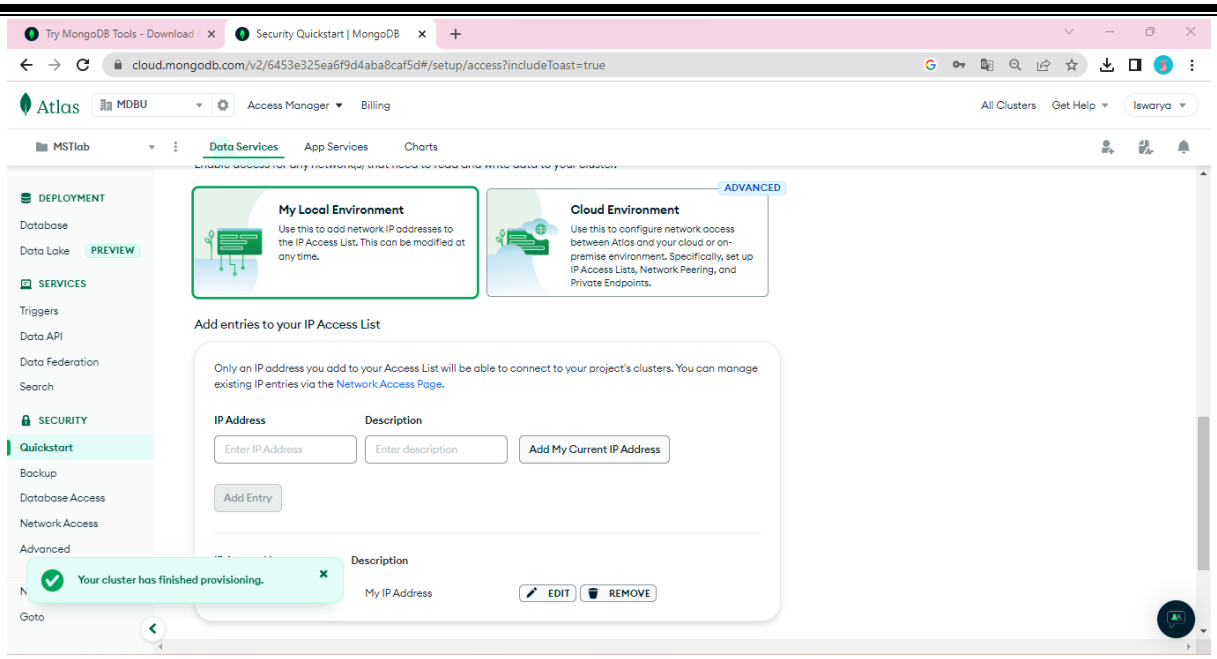




Click on create project

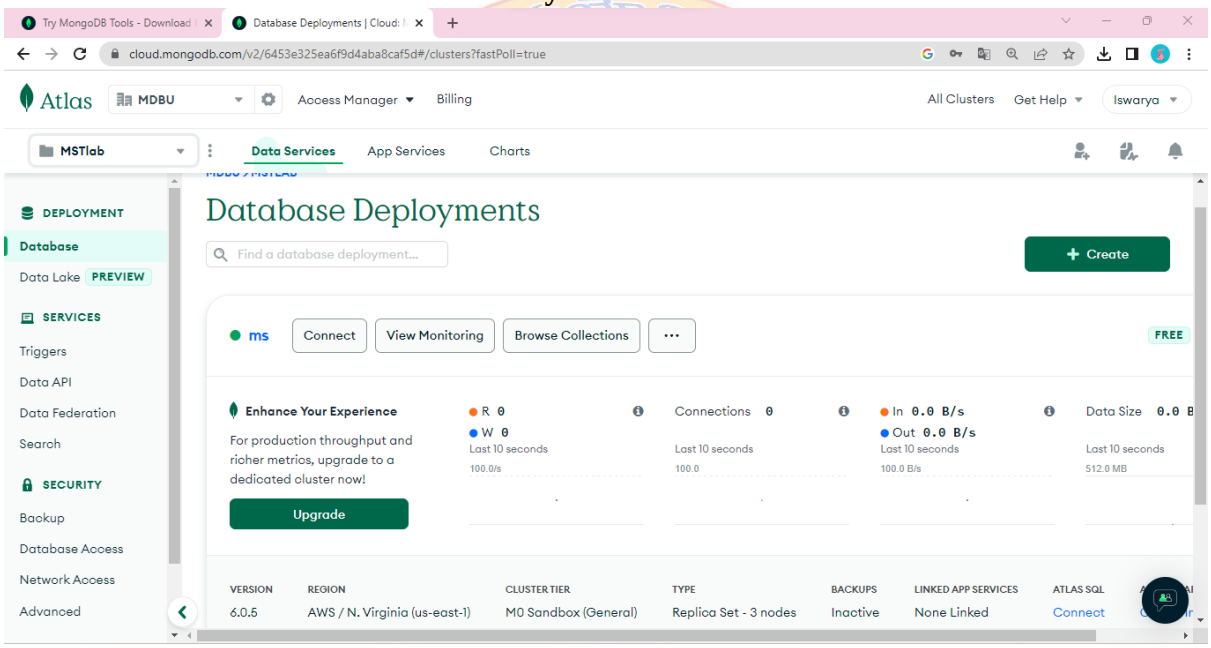



Try MongoDB Tools - Download
Deploy your database | Cloud: M
cloud.mongodb.com/v2/6453e325ea6f9d4aba8caf5d#/clusters/starterTemplates
MongoDB.
Deploy your database
Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.
M10 $0.08/hour
For production applications with sophisticated workload requirements.
STORAGE 10 GB RAM 2 GB vCPU 2 vCPUs
SERVERLESS $0.10/1M reads
For application development and testing, or workloads with variable traffic.
STORAGE Up to 1 TB RAM Auto-scale vCPU Auto-scale
M0 FREE
For learning and exploring MongoDB in a cloud environment.
STORAGE 512 MB RAM Shared vCPU Shared
FREE
Create
Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.
I'll deploy my database later
Access Advanced Configuration
Provider
aws
Google Cloud
Azure
Region
Recommended region
N. Virginia (us-east-1)
Name
You cannot change the name once the cluster is created.
ms
FREE
Create
Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.
I'll deploy my database later
Access Advanced Configuration
Security Quickstart | MongoDB
cloud.mongodb.com/v2/6453e325ea6f9d4aba8caf5d#/setup/access?includeToast=true
Atlas
MDBU
Access Manager
Billing
All Clusters
Get Help
Iswarya
MSTlab
Data Services
App Services
Charts
DEPLOYMENT
Database
Data Lake PREVIEW
SERVICES
Triggers
Data API
Data Federation
Search
SECURITY
Quickstart
Backup
Advanced
New On Atlas
Create a database user using a username and password. Users will be given the read and write to any database privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password. You can manage existing users via the Database Access Page.
Username
durga
Password
Autogenerate Secure Password
Copy
Create User
This password contains special characters which will be URL-encoded.
Your cluster has finished provisioning.
Authentication Type
ms
Password
EDIT
REMOVE



Change the password if required and click on add my current IP Address, select Finish and close and go to databases.

So now the cluster is created successfully.



Select the browse collections to add databases and collections.

11.b Course Name: MongoDB Essentials - A Complete MongoDB Guide Module Name: Introduction to the CRUD Operations

Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()

Aim: To Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove().

Description:

MongoDB is a persistent document-oriented database used to store and process data in the form of documents. As with other database management systems, MongoDB allows you to manage and interact with data through four fundamental types of data operations:

- Create operations, which involve writing data to the database
- Read operations, which query a database to retrieve data from it
- Update operations, which change data that already exists in a database
- Delete operations, which permanently remove data from a database

These four operations are jointly referred to as *CRUD* operations.

Queries:

```
> show databases
< admin    40.00 KiB
  config  108.00 KiB
  local    88.00 KiB
User>
```

```
> use ms
<   'switched to db ms'
ms>
> show dbs
< admin    40.00 KiB
  config  108.00 KiB
  local    88.00 KiB
  ms        8.00 KiB
```

```
> use sample
<   'switched to db sample'
> db.createCollection("details")
<   { ok: 1 }
> show collections
< details
sample>
```

```
> db.createCollection("first")
<   { ok: 1 }
> show collections
< details
  first
sample >
```

Insert:
To insert one document

```
> db.details.insertOne({"name":"James","mobile":"1234567890","mail":"james123@gmail.com"})
<   {
      acknowledged: true,
      insertedId: ObjectId("64599f27932c399f1b122c7e")
    }
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
sample >
```

To insert many documents:
Syntax:db.collectionname.insertMany([{..},{…}])

```
  }
> db.details.insertMany([{"name":"john","mobile":"9087654321","mail":"john@gmail.com"},{"name":"riya","mobile":"7890654321","username":"riya_06","mail":"riya@gmail.com"}])
<  {
     acknowledged: true,
     insertedIds: {
       '0': ObjectId("6459a142932c399f1b122c7f"),
       '1': ObjectId("6459a142932c399f1b122c80")
     }
   }
```

```
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459a142932c399f1b122c7f"),
      name: 'john',
      mobile: '9087654321',
      mail: 'john@gmail.com'
    }
    {
      _id: ObjectId("6459a142932c399f1b122c80"),
      name: 'riya',
      mobile: '7890654321',
      username: 'riya_06',
      mail: 'riya@gmail.com'
    }
sample>
```

Find:

```
> db.details.find({"username":"riya_06"})
<   {
      _id: ObjectId("6459a142932c399f1b122c80"),
      name: 'riya',
      mobile: '7890654321',
      username: 'riya_06',
      mail: 'riya@gmail.com'
    }
```

Update:

To update one document:

```
> db.details.updateOne({"name":"john"},{$set:{"mobile":"9898989876"}})
<   {
      acknowledged: true,
      insertedId: null,
      matchedCount: 1,
      modifiedCount: 1,
      upsertedCount: 0
    }
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459a142932c399f1b122c7f"),
      name: 'john',
      mobile: '9898989876',
      mail: 'john@gmail.com'
    }
    {
      _id: ObjectId("6459a142932c399f1b122c80"),
      name: 'riya',
      mobile: '7890654321',
      username: 'riya_06',
      mail: 'riya@gmail.com'
    }
```

So here only one document with name John has been updated.

To update many documents:

```
> db.details.updateMany({"name":"john"},{$set:{"mail":"john456@gmail.com"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
```

```
  }
  {
    _id: ObjectId("6459a142932c399f1b122c7f"),
    name: 'john',
    mobile: '9898989876',
    mail: 'john456@gmail.com'
  }
  {
    _id: ObjectId("6459a142932c399f1b122c80"),
    name: 'riya',
    mobile: '7890654321',
    username: 'riya_06',
    mail: 'riya@gmail.com'
  }
  {
    _id: ObjectId("6459c82f048bc3ee8b100c1d"),
    name: 'john',
    mobile: '7345612890',
    email: 'john12@gmail.com',
    mail: 'john456@gmail.com'
  }
```

Here both the documents with name John has been updated.

Delete:

To delete one document:

```
> db.details.deleteOne({"name":"riya"})
<   {
      acknowledged: true,
      deletedCount: 1
    }
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459a142932c399f1b122c7f"),
      name: 'john',
      mobile: '9898989876',
      mail: 'john456@gmail.com'
    }
    {
      _id: ObjectId("6459c82f048bc3ee8b100c1d"),
      name: 'john',
      mobile: '7345612890',
      email: 'john12@gmail.com',
      mail: 'john456@gmail.com'
    }
```

To delete Many documents:

```
> db.details.deleteMany({"name":"john"})
<   {
      acknowledged: true,
      deletedCount: 2
    }
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
sample >
```

12.a Course Name: MongoDB Essentials - A Complete MongoDB Guide
Module Name: Create and Delete Databases and Collections
Write MongoDB queries to Create and drop databases and collections.

AIM:

To Write MongoDB queries to Create and drop databases and collections.

DESCRIPTION:

MongoDB stores data records as documents (specifically BSON documents) which are gathered together in collections. A database stores one or more collections of documents.

Databases:

If a database does not exist, MongoDB creates the database when you first store data for that database.

In MongoDB, databases hold one or more collections of documents.

Collections:

MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

If a collection does not exist, MongoDB creates the collection when you first store data for that collection.

Queries:

To create Database:

```
>_MONGOSH
> use User
<   'switched to db User'
User>
```

To create Collection:

```
> db.createCollection("userdata")
<   { ok: 1 }
User>
> show collections
< userdata
User>
```

To drop collections:

```
> db.userdata.drop()
<   true
> show collections
<
User>
```

To drop Database:
db.dropDatabase()

```
> db.dropDatabase()
<   { ok: 1, dropped: 'User' }
User>
```

```
> show databases
< admin    40.00 KiB
  config  108.00 KiB
  local    88.00 KiB
User>
```

12.b Course Name: MongoDB Essentials - A Complete MongoDB Guide
Module Name: Introduction to MongoDB Queries
Write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

AIM:To write MongoDB queries to work with records using find(), limit(), sort(), createIndex(), aggregate().

Find():

```
> db.details.find()
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459cb4c048bc3ee8b100c1e"),
      name: 'john',
      mobile: '7345612890',
      email: 'john12@gmail.com'
    }
    {
      _id: ObjectId("6459cb6e048bc3ee8b100c1f"),
      name: 'joy',
      mobile: '7345555890',
      email: 'joy12@gmail.com'
    }
    {
      _id: ObjectId("6459cbbd048bc3ee8b100c20"),
      name: 'aria',
      mobile: '8445555890',
      email: 'aria@gmail.com'
    }
```

To get only particular fields:

Syntax: db.collectionname.find({query},{filedname1:1,fieldname2:1,..})

1 indicate to show the field and 0 indicates not to show the field.

```
> db.details.find({},{name:1})
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James'
    }
    {
      _id: ObjectId("6459cb4c048bc3ee8b100c1e"),
      name: 'john'
    }
    {
      _id: ObjectId("6459cb6e048bc3ee8b100c1f"),
      name: 'joy'
    }
    {
      _id: ObjectId("6459cbbd048bc3ee8b100c20"),
      name: 'aria'
    }
```

Limit():

In MongoDB, the limit() method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want.

Syntax: db.collectionName.find(<query>).limit(<number>)

```
> db.details.find().limit(2)
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459cb4c048bc3ee8b100c1e"),
      name: 'john',
      mobile: '7345612890',
      email: 'john12@gmail.com'
    }
```

Sort():

The sort() method specifies the order in which the query returns the matching documents from the given collection.

The value is 1 or -1 specify an ascending or descending sort respectively.

Syntax: db.Collection_name.sort({field_name:1 or -1})

Before:

```
> db.details.find({},{name:1})
< {
    _id: ObjectId("64599f27932c399f1b122c7e"),
    name: 'James'
  }
  {
    _id: ObjectId("6459cb4c048bc3ee8b100c1e"),
    name: 'john'
  }
  {
    _id: ObjectId("6459cb6e048bc3ee8b100c1f"),
    name: 'joy'
  }
  {
    _id: ObjectId("6459cbbd048bc3ee8b100c20"),
    name: 'aria'
  }
```

After sorting:

```
> db.details.find().sort({"name":1})
<   {
      _id: ObjectId("64599f27932c399f1b122c7e"),
      name: 'James',
      mobile: '1234567890',
      mail: 'james123@gmail.com'
    }
    {
      _id: ObjectId("6459cbbd048bc3ee8b100c20"),
      name: 'aria',
      mobile: '8445555890',
      email: 'aria@gmail.com'
    }
    {
      _id: ObjectId("6459cb4c048bc3ee8b100c1e"),
      name: 'john',
      mobile: '7345612890',
      email: 'john12@gmail.com'
    }
    {
      _id: ObjectId("6459cb6e048bc3ee8b100c1f"),
      name: 'joy',
      mobile: '7345555890',
      email: 'joy12@gmail.com'
    }
sample >
```

CreateIndex():

Creating an Index in MongoDB is done by using the "**createIndex**" method.

If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.

```
> db.details.find({"name":"aria"}).explain('executionStats')
```

```
executionStats: {
  executionSuccess: true,
  nReturned: 1,
  executionTimeMillis: 22,
  totalKeysExamined: 0,
  totalDocsExamined: 4,
  executionStages: {
    stage: 'COLLSCAN',
    filter: {
      name: {
        '$eq': 'aria'
      }
    },
```

Here the documents scanned are 5.

```
> db.details.createIndex({name:1})
<   'name_1'
> db.details.find({"name":"aria"}).explain('executionStats')
```

```
executionStats: {
  executionSuccess: true,
  nReturned: 1,
  executionTimeMillis: 35,
  totalKeysExamined: 1,
  totalDocsExamined: 1,
  executionStages: {
    stage: 'FETCH',
    nReturned: 1,
    executionTimeMillisEstimate: 10,
    works: 2,
    advanced: 1,
    needTime: 0,
    needYield: 0,
    saveState: 0,
    restoreState: 0,
    isEOF: 1,
    docsExamined: 1,
    alreadyHasObj: 0,
    inputStage: {
      stage: 'IXSCAN',
```

Here only one document is scanned.

aggregate():

In MongoDB, aggregate methods are used to perform complex data analysis tasks on collections of data. These methods allow you to process and transform data in a variety of ways, such as grouping, sorting, filtering, and computing statistical calculations.
Here are some of the commonly used aggregate methods in MongoDB:
$group - Groups documents in a collection by a specified key and applies aggregate functions to the grouped data.
$match - Filters documents in a collection based on a specified condition.
$sort - Sorts documents in a collection based on a specified order.
$project - Specifies which fields to include or exclude in the output document.
$limit - Limits the number of documents returned by an aggregate operation.
$skip - Skips a specified number of documents in an aggregate operation.
$unwind - Separates arrays in a document into separate documents.
$lookup - Performs a left outer join between two collections.
$sum - Calculates the sum of values in a field.
$avg - Calculates the average of values in a field.
$max - Finds the maximum value of a field.
$min - Finds the minimum value of a field.
These aggregate methods can be combined and used in various ways to achieve complex data analysis tasks in MongoDB.

```
> db.details.aggregate([
    {
      $group: {
        _id: "$branch",
        count: { $sum: 1 },
        students: { $push: "$name" }
      }
    }
    ])
```

```
<   {
      _id: null,
      count: 4,
      students: [
        'James',
        'john',
        'joy',
        'aria'
      ]
    }
```