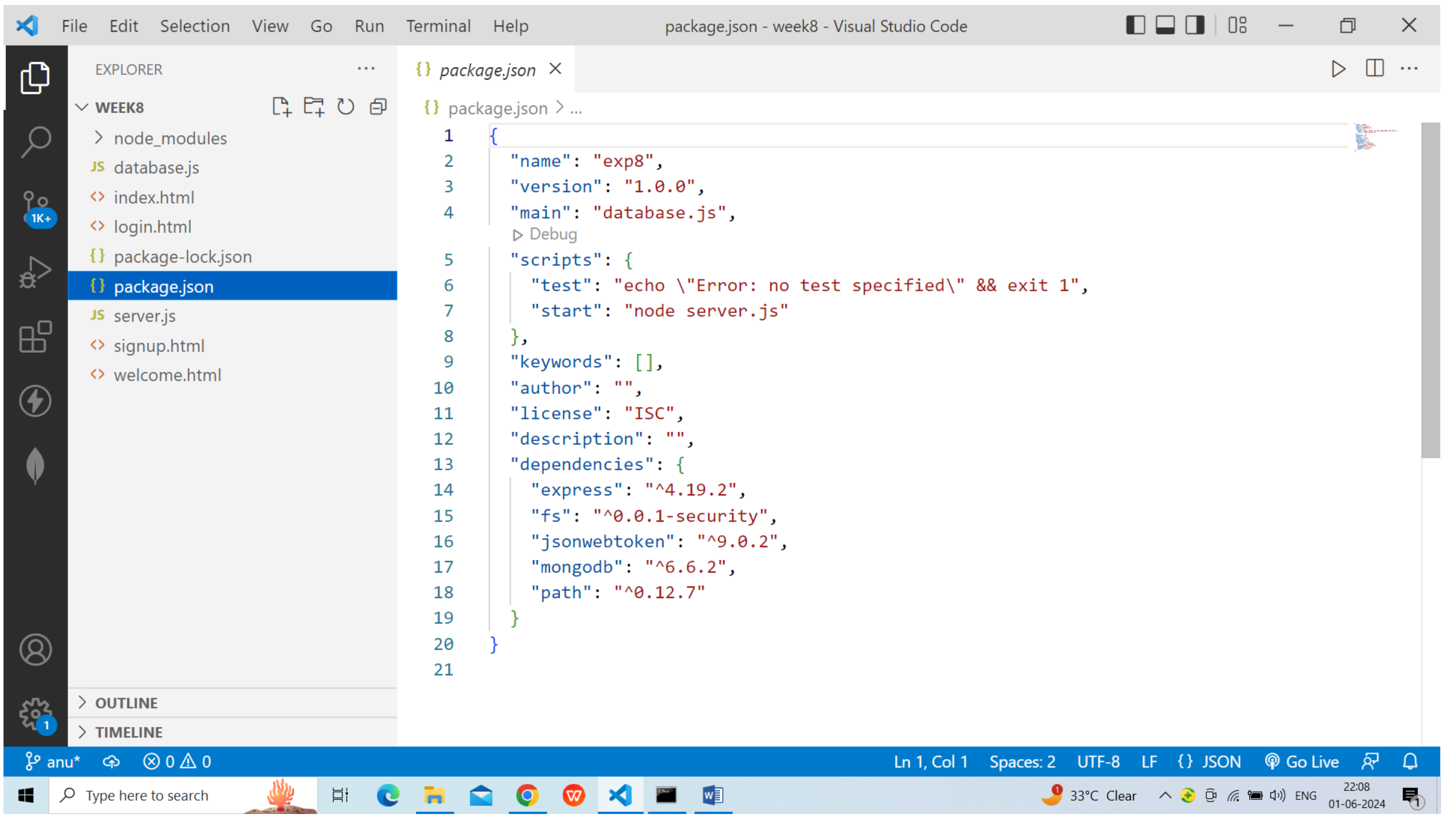
Week 8

Write MongoDB model for the user and query to fetch user, validate and register.



```
{
  "name": "exp8",
  "version": "1.0.0",
  "main": "database.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "express": "^4.19.2",
    "fs": "^0.0.1-security",
    "jsonwebtoken": "^9.0.2",
    "mongodb": "^6.6.2",
    "path": "^0.12.7"
  }
}
```

Softwares:-Node Js 20 version, visual studio and Postman

Steps:-Create package.json file initializing project using command npm(node package manager)

- **npm init (or) npm init –y**
  package.json file created show above fig.1.
- Installing require modules
  npm i express nodemon fs path jsonwebtoken mongodb

Note:
Download Postman or thunder client for middleware.

Database.js:-

```
const {MongoClient} =require('mongodb');
const url = 'mongodb://127.0.0.1:27017';
const client= new MongoClient(url);
let collection;
//can also export like as follows
```

```
// module.exports={getData,insertData};

async function connectDB(dbname,table)
{
    let result=await client.connect();
    let db= result.db(dbname);
    collection = db.collection(table);
    console.log("DataBase Connected...");
    return collection;
}

exports.getData=async function (name,password)
{
   collection = await connectDB("cse","employee");
   let response = await collection.find({name:name,password:password}).toArray();
   console.log("From getData method: "+JSON.stringify(response));
   collection.close;
   return JSON.stringify(response);
}

exports.insertData=async function (emp)
{
    collection = await connectDB("cse","employee");
    let response=await
collection.insertOne({name:emp.name,work:emp.work,password:emp.password})
    console.log("Record inserted Successfully");
    collection.close;
    return JSON.stringify(response);
}
```

Login.html:-

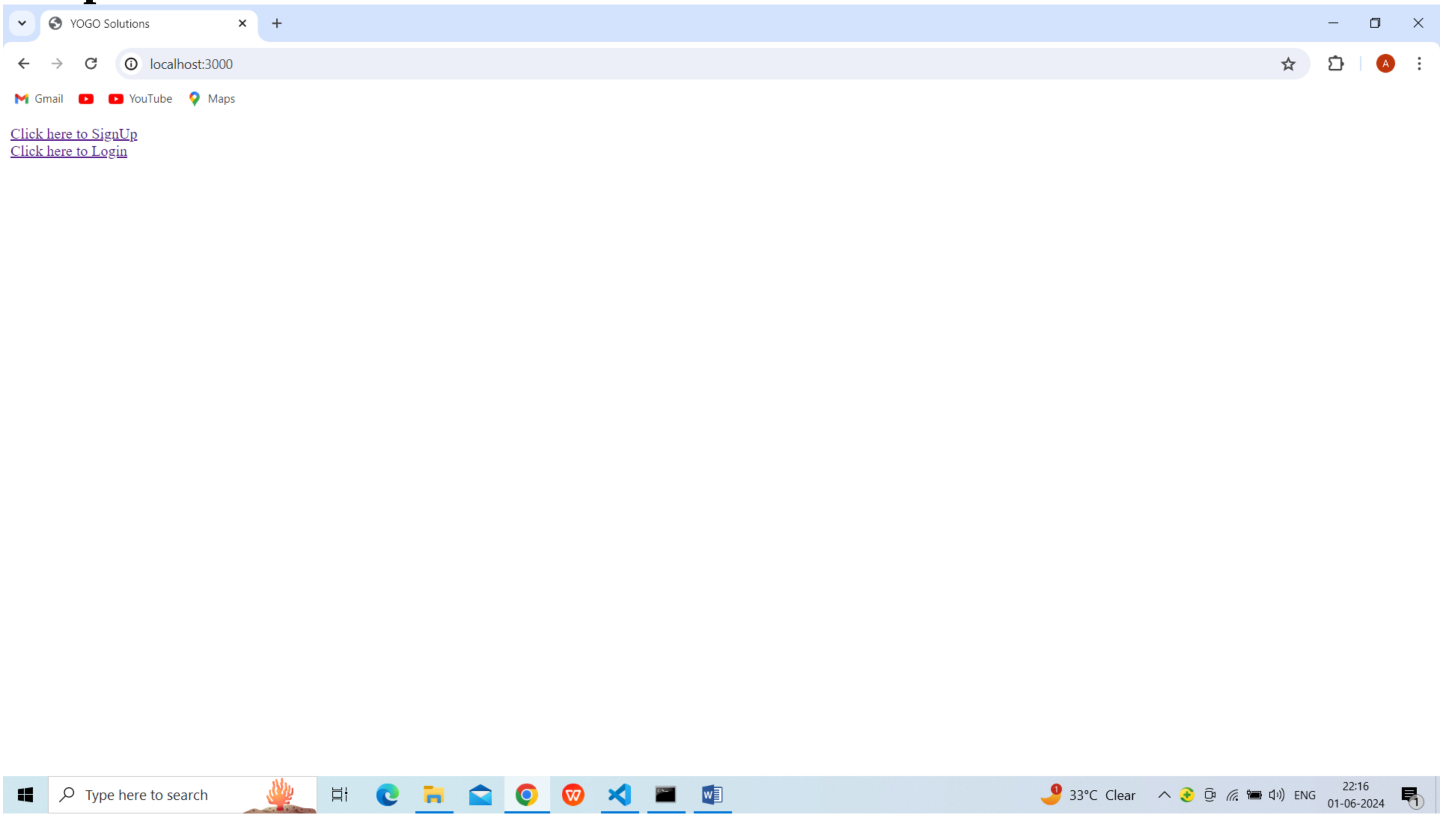
```
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
const form=$("#loginForm");
$(document).ready(function() {
$("#loginForm").submit((event)=>{
event.preventDefault();
var name = $("#username").val();
var password=$("#pwd").val();
//generateToken by making a service call to /auth with post
$.ajax({
url:"/auth",
type:"POST",
contentType:"application/json",
dataType:"json",
data: JSON.stringify({
name: name,
password: password
}),
success: function(data)
{
var tokenData=data;
if(tokenData.login==true){
            console.log("token data: "+JSON.stringify(tokenData));
            localStorage.setItem("username",tokenData.data[0].name);
            localStorage.setItem("job",tokenData.data[0].work);
            if(verifyLogin(tokenData.token))
            {
              window.location.href="/welcome";
            }
            else{
              alert("Authentication Failed");
            }
          }
          },
          error: function(data ){
            console.log("Something went wrong");
            }
          });
        });
});

function verifyLogin(token)
{
  let result=true;
  //verifyToken by making a service call to /verifyToken with get
  $.ajax({
    url:"/verifyToken",
    type:"POST",
    contentType:"application/json",
    dataType:"json",
    data: JSON.stringify({
    token: token,
    }),
    success: function(data) {
      if(data.login==true)
      {
          result=true;
      }
      else
      {
        result=false;
      }
   },
          error: function(data ){
            console.log("Wrong Token, Not Authenticated.");
            }
          });
    return result;
}
  </script>
<body>
<h2>Sign In</h2>
```

```html
<form id="loginForm">
  <label for="username">Username:</label><br>
  <input type="text" id="username" name="username"><br>
  <label for="pwd">Password:</label><br>
  <input type="password" id="pwd" name="pwd"><br><br>
  <input type="submit" value="Submit">
</form>
</body>
</html>
```

Server.js:-

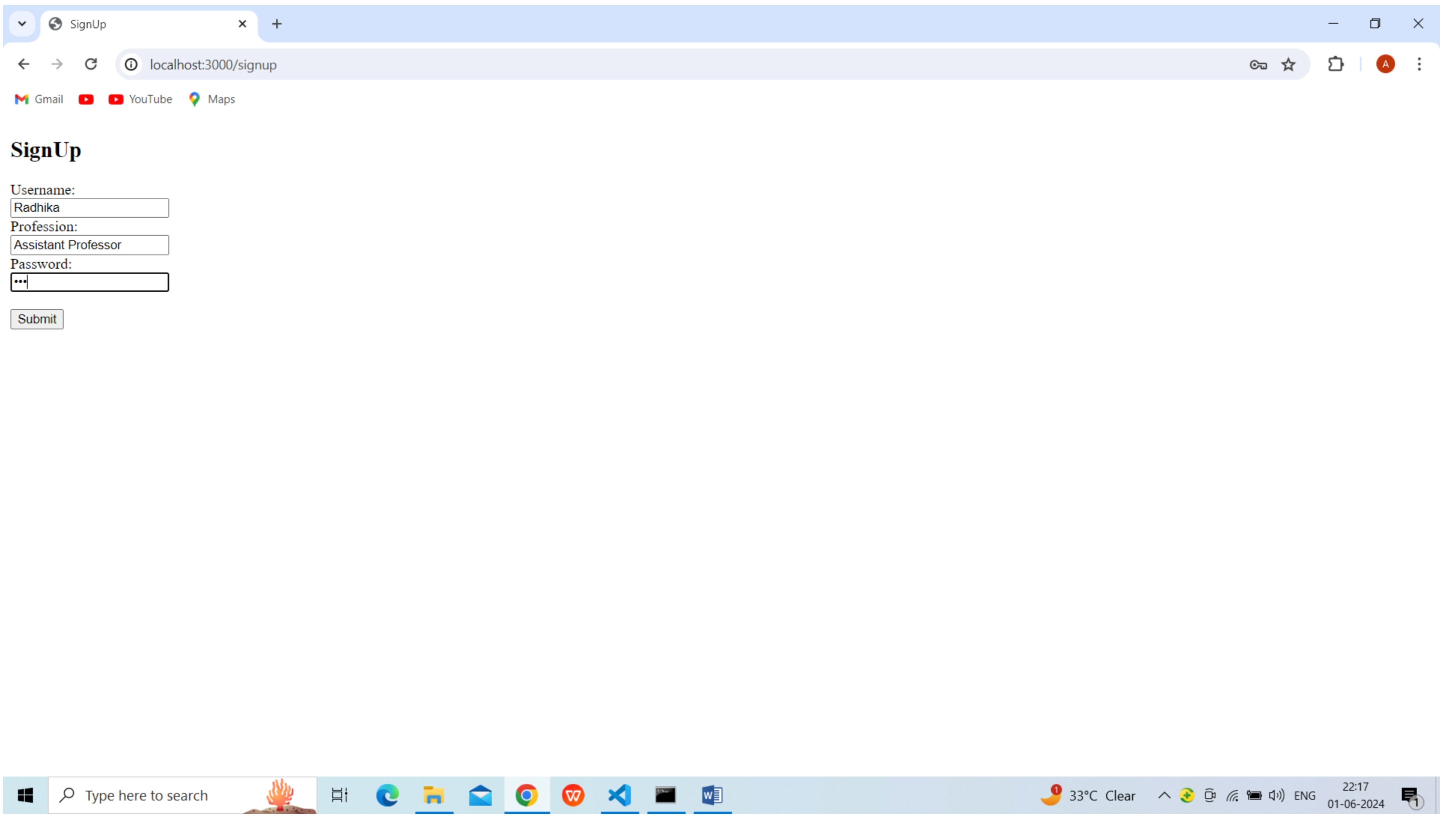
```javascript
// Import express for creating API's endpoints
const express = require("express");
const db = require("./database.js");
const {MongoClient} =require('mongodb');
var token;

// Import jwt for API's endpoints authentication
const jwt = require("jsonwebtoken");

// Creates an Express application, initiate
// express top level function
const app = express();

// A port for serving API's
const port = 3000;

// Allow json data
app.use(express.json());

app.get('/',
    (req, res) => {
        res.sendFile(__dirname + '/index.html');
    });
app.get('/welcome',
    (req, res) => {
        res.sendFile(__dirname + '/welcome.html');
    });
app.get('/signup',
    (req, res) => {
        res.sendFile(__dirname + '/signup.html');
    });
app.get('/login',
    (req, res) => {
        res.sendFile(__dirname + '/login.html');
    });

// SignUp route
app.post("/register", async(req, res) => {
    // Get the name to the json body data
    const name = req.body.name;

    // Get the password to the json body data
    const password = req.body.password;

    // Get the profession to the json body data
    const work = req.body.work;
    tuple= await db.getData(name,password);
        if(JSON.parse(tuple).length>0){
        res.json({
            signup: false,
            token: null,
            error: "Already registered",
        });
    } else {

        // The jwt.sign method are used
        // to create token
        const token = jwt.sign(tuple, "secret");

    let emp =
    {
        name: name,
        work: work,
        password: password,
        token:token
    };

    let result=await db.insertData(emp);
    res.json({
        signup: true,
        token: "generated",
        result:result
    });
    }
});


// Login route
app.post("/auth", async (req, res) => {
    // Get the name to the json body data

    const name = req.body.name;
    console.log(name);

    // Get the password to the json body data
    const password = req.body.password;
    console.log(password)

    tuple=await db.getData(name,password);

    if (JSON.parse(tuple).length>0) {

        // The jwt.sign method are used
        // to create token
        const token = jwt.sign(tuple, "secret");

        // Pass the data or token in response
        res.json({
            login: true,
            token: token,
            data: JSON.parse(tuple),
        });
    } else {
        res.json({
            login: false,
            error: "please check name and password.",
        });
    }
});

// Verify route
app.post("/verifyToken", (req, res) => {

    // Get token value to the json body
    const token = req.body.token;

    // If the token is present
    if (token) {

        // Verify the token using jwt.verify method
        const decode = jwt.verify(token, "secret");

        // Return response with decode data
        res.json({
            login: true,
            data: decode,
        });
    } else {

        // Return response with error
        res.json({
            login: false,
            data: "error",
        });
    }
});

app.post('/welcome',(req, res) => {
    res.redirect("/welcome")
    });
app.post('/login',(req, res) => {
        res.redirect("/login")
        });
app.post('/signup',(req, res) => {
    res.redirect("/signup")
    });

// Listen the server
app.listen(port, () => {
    console.log(`Server is running :
    http://localhost:${port}/`);
});
```

Output:-



Click here to SignUp
Click here to Login

Fig.1. Index Page

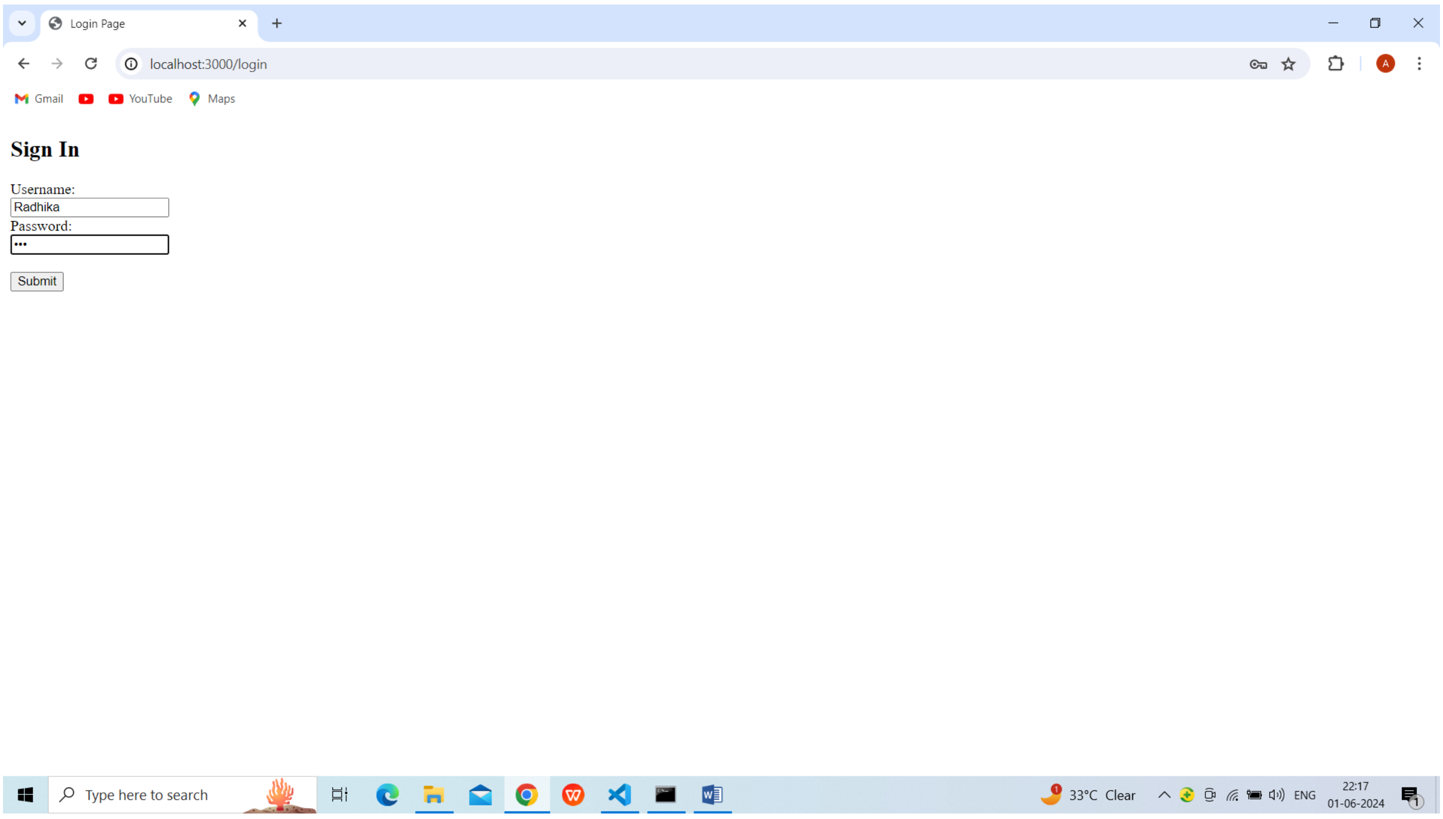


SignUp

Username: Radhika
Profession: Assistant Professor
Password: •••

Submit

Fig.2. SignUp Page


Login Page

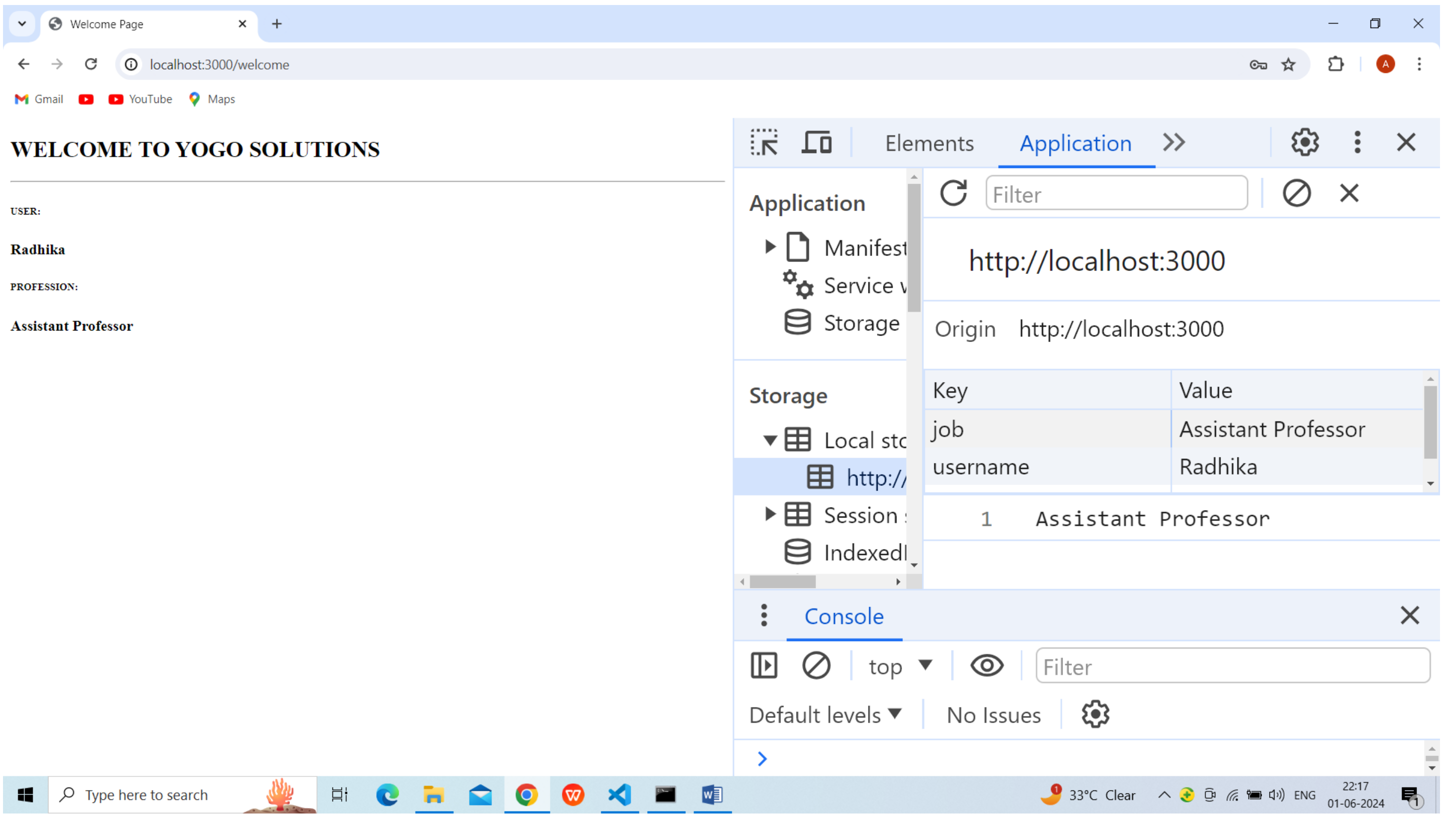
localhost:3000/login

Sign In

Username: Radhika
Password: •••

Submit

Fig.3. SignIn Page


Welcome Page

localhost:3000/welcome

WELCOME TO YOGO SOLUTIONS

USER:

Radhika

PROFESSION:

Assistant Professor

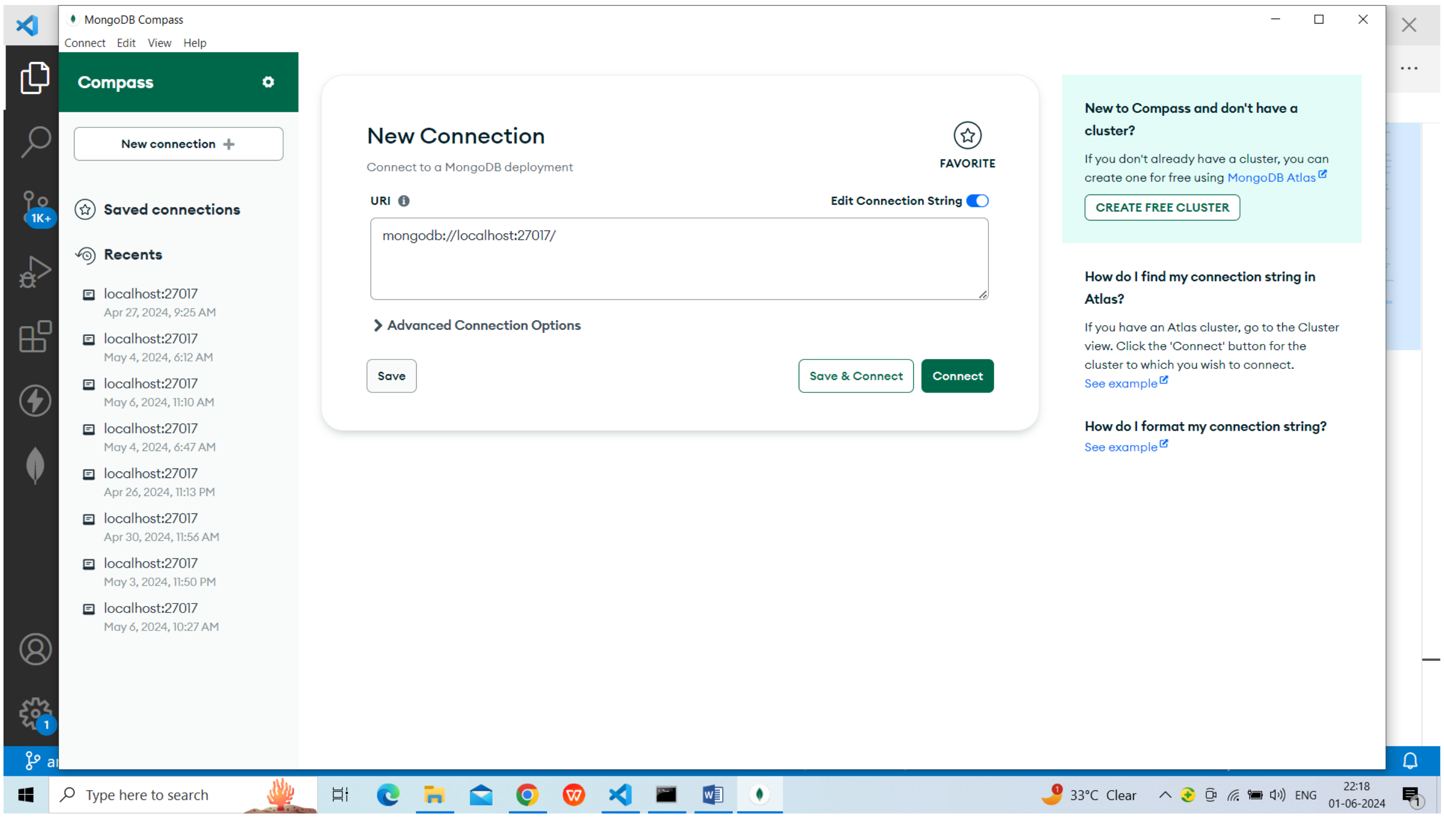
http://localhost:3000

Origin http://localhost:3000

Key	Value
job	Assistant Professor
username	Radhika

```
1  Assistant Professor
```

Fig.4. Welcome Page


MongoDB Compass

Connect Edit View Help

Compass

New connection +

Saved connections

Recents

localhost:27017 Apr 27, 2024, 9:25 AM

localhost:27017 May 4, 2024, 6:12 AM

localhost:27017 May 6, 2024, 11:10 AM

localhost:27017 May 4, 2024, 6:47 AM

localhost:27017 Apr 26, 2024, 11:13 PM

localhost:27017 Apr 30, 2024, 11:56 AM

localhost:27017 May 3, 2024, 11:50 PM

localhost:27017 May 6, 2024, 10:27 AM

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

How do I find my connection string in Atlas?

If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.

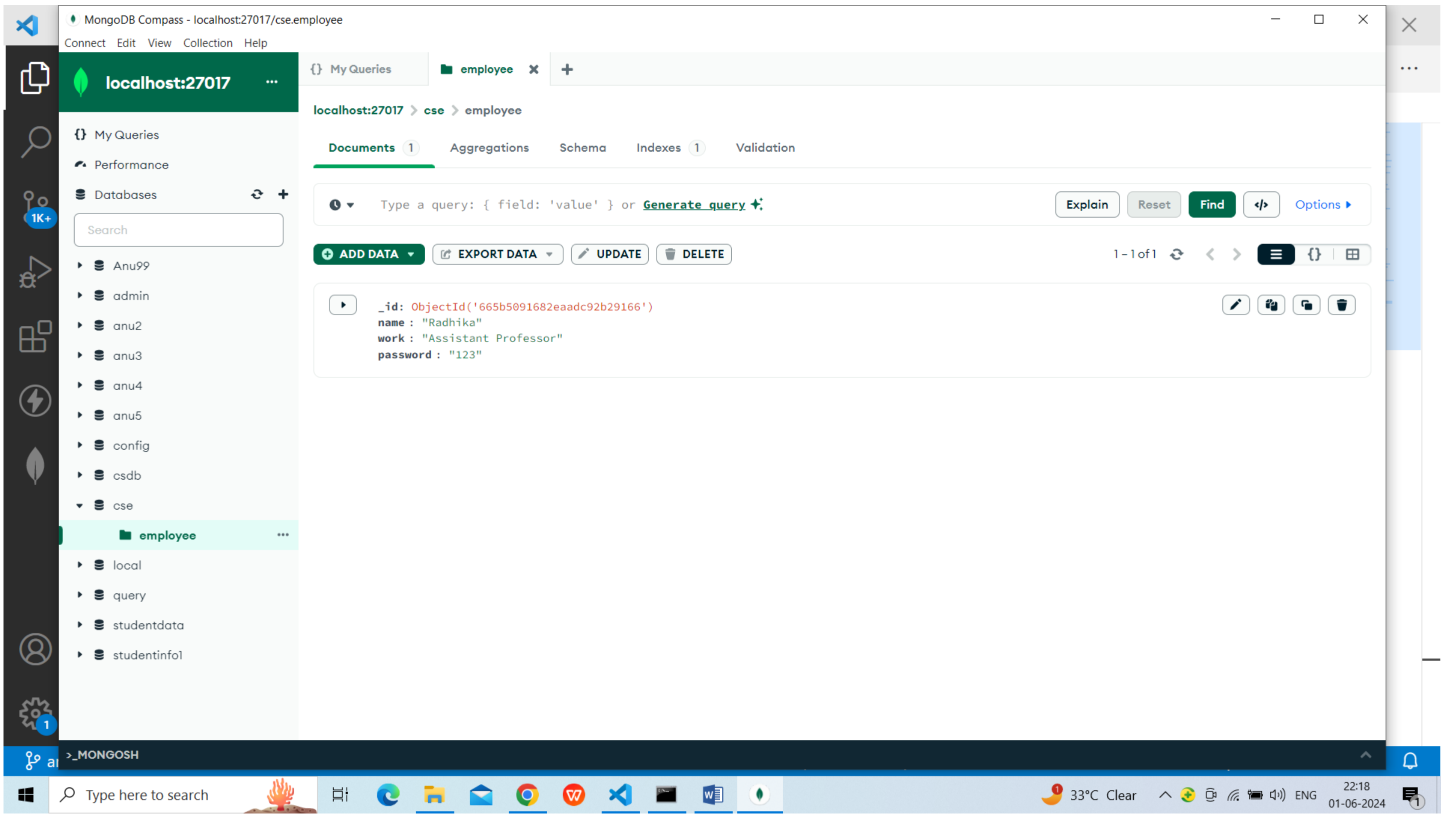
See example

How do I format my connection string?

See example

Type here to search 33°C Clear ENG 22:18 01-06-2024

Fig.5. MongoDB Compass Connection Page


MongoDB Compass - localhost:27017/cse.employee

Connect Edit View Collection Help

localhost:27017

My Queries

Performance

Databases

Search

Anu99, admin, anu2, anu3, anu4, anu5, config, csdb, cse, employee, local, query, studentdata, studentinfo1

My Queries employee

localhost:27017 > cse > employee

Documents 1 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or Generate query

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 1 - 1 of 1

```
_id: ObjectId('665b5091682eaadc92b29166')
name : "Radhika"
work : "Assistant Professor"
password : "123"
```

>_MONGOSH

Type here to search 33°C Clear ENG 22:18 01-06-2024

Fig.6. After SignUp Data added to Mongodb database Page

VIVA QUESTIONS:

1.What type of DBMS is MongoDB?

Ans: MongoDB is a NoSQL database management system (DBMS) known for its scalability, flexibility, and performance. Unlike traditional relational database management systems (RDBMS) that store data in tables with rows and columns, MongoDB uses a document-oriented approach to store data.

Key Characteristics of MongoDB:
1.Document-Oriented Storage
2.Schema-less
3.Scalability
4.Querying and Indexing
5.Aggregation Framework
6.NoSQL Database

2.Difference between MySQL and MongoDB ?

Ans: MySQL: Uses a relational model where data is stored in tables with rows and columns. Relations among data are defined through foreign keys. It follows a strict schema, requiring the structure of the data (tables and columns) to be defined beforehand.
MongoDB: Utilizes a document model, storing data in BSON (binary-encoded JSON) documents. This allows for more flexible data structures, with the ability to store nested data. MongoDB is schema-less, meaning the documents in a collection can have different structures.

MySQL: Offers strong consistency and supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring reliable processing of transactions even in complex operations involving multiple tables.
MongoDB: Initially favored eventual consistency to provide high availability and partition tolerance but has since introduced support for ACID transactions since version 4.0. However, its transaction support is generally considered more limited compared to traditional RDBMS.

3.Which technologies can support multiple databases ?

Ans: 1.ORM (Object-Relational Mapping) Libraries ORM libraries allow developers to interact with a database using an object-oriented paradigm, abstracting the underlying database interactions.

Hibernate (Java): A robust ORM framework that supports a wide range of databases including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and more.
Entity Framework (C#/.NET): Microsoft's ORM for .NET applications, supporting SQL Server, MySQL, PostgreSQL, SQLite, and more through various providers.
Sequelize (Node.js): A promise-based ORM for Node.js, supporting PostgreSQL, MySQL, MariaDB, SQLite, and Microsoft SQL Server.
Django ORM (Python): Part of the Django web framework, it supports PostgreSQL, MySQL,SQLite, and Oracle Database.
SQLAlchemy (Python): A SQL toolkit and ORM that provides full power and flexibility of SQL to Python applications, supporting numerous databases including PostgreSQLMySQL, and SQLite.

4.How do we perform the join operations in MongoDB ?

Ans: In MongoDB, join operations are primarily performed using the $lookup stage in an aggregation pipeline. MongoDB's aggregation framework is a powerful tool for data transformation and analysis, allowing for complex queries and operations on the data, including the capability to perform operations similar to SQL joins. The $lookup stage lets you pull documents from another collection (table in SQL terminology) into your aggregation query, effectively performing a "join" between two collections.

5.Explain indexes in MongoDB?

Ans: Indexes in MongoDB are special data structures that store a small portion of the collection's data in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index. They are used to improve the speed of search operations within MongoDB collections by allowing the database engine to efficiently find documents matching the query criteria without scanning every document in a collection.

1.Single Field Index
2.Compound Index
3.Multikey Index
4.Text Index
5.Hashed Index
6.Geospatial Index
7.Unique Index
8.Partial Index