Documentation for Apprenticeships service

**1. Setting Up Infrastructure with Bicep on Azure**

**a. Install Bicep CLI**

First, ensure you have Bicep installed on your machine:

``` sh

Copy code

az bicep install

**b. Create a Bicep Configuration**

Create a directory for your Bicep files and add the following Bicep configuration:

```

C:\Users\James\_Cole>mkdir bicepfiles

C:\Users\James\_Cole>cd bicepfiles

C:\Users\James\_Cole\bicepfiles>

```

**main.bicep**:

bicep

```

// Define the parameters

param location string = 'UK South'

// Define the storage account

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-09-01' = {

name: 'jobsitestoracc'

location: location

sku: {

name: 'Standard\_LRS'

}

kind: 'StorageV2'

}

// Define the App Service plan

resource appServicePlan 'Microsoft.Web/serverfarms@2021-03-01' = {

name: 'jobsite-asp'

location: location

sku: {

tier: 'Standard'

size: 'S1'

}

}

// Define the web app

resource webApp 'Microsoft.Web/sites@2021-02-01' = {

name: 'jobsite-app'

location: location

properties: {

serverFarmId: appServicePlan.id

siteConfig: {

appSettings: [

{

name: 'WEBSITE\_RUN\_FROM\_PACKAGE'

value: '1'

}

]

}

}

}

// Define the SQL Server

resource sqlServer 'Microsoft.Sql/servers@2021-02-01-preview' = {

name: 'jobsitesqlserver'

location: location

properties: {

administratorLogin: 'sqladmin'

administratorLoginPassword: 'Password123!' // Use a secure password in practice

}

}

// Define the SQL database

resource sqlDatabase 'Microsoft.Sql/servers/databases@2021-02-01-preview' = {

name: 'jobsite-db'

parent: sqlServer

properties: {

collation: 'SQL\_Latin1\_General\_CP1\_CI\_AS'

maxSizeBytes: 2147483648

}

sku: {

name: 'S0'

}

}

// Output the web app URL

output webAppUrl string = 'https://${webApp.properties.defaultHostName}'```

**c. Deploy Bicep Configuration**

Use Azure CLI to deploy the Bicep file:

Sh

**Create the group of resources**

```

az group create --name jobsite-rg --location "UK South"

```

**Test it for errors**

```

az bicep build --file main.bicep

```

**Deploy the group**

```

az deployment group create --resource-group jobsite-rg --template-file main.bicep --parameters location="UK South"```

**Delete when no longer needed**

```

az group delete --name jobsite-rg --yes --no-wait

```

**2.0 Setting Up SQL Server and Database**

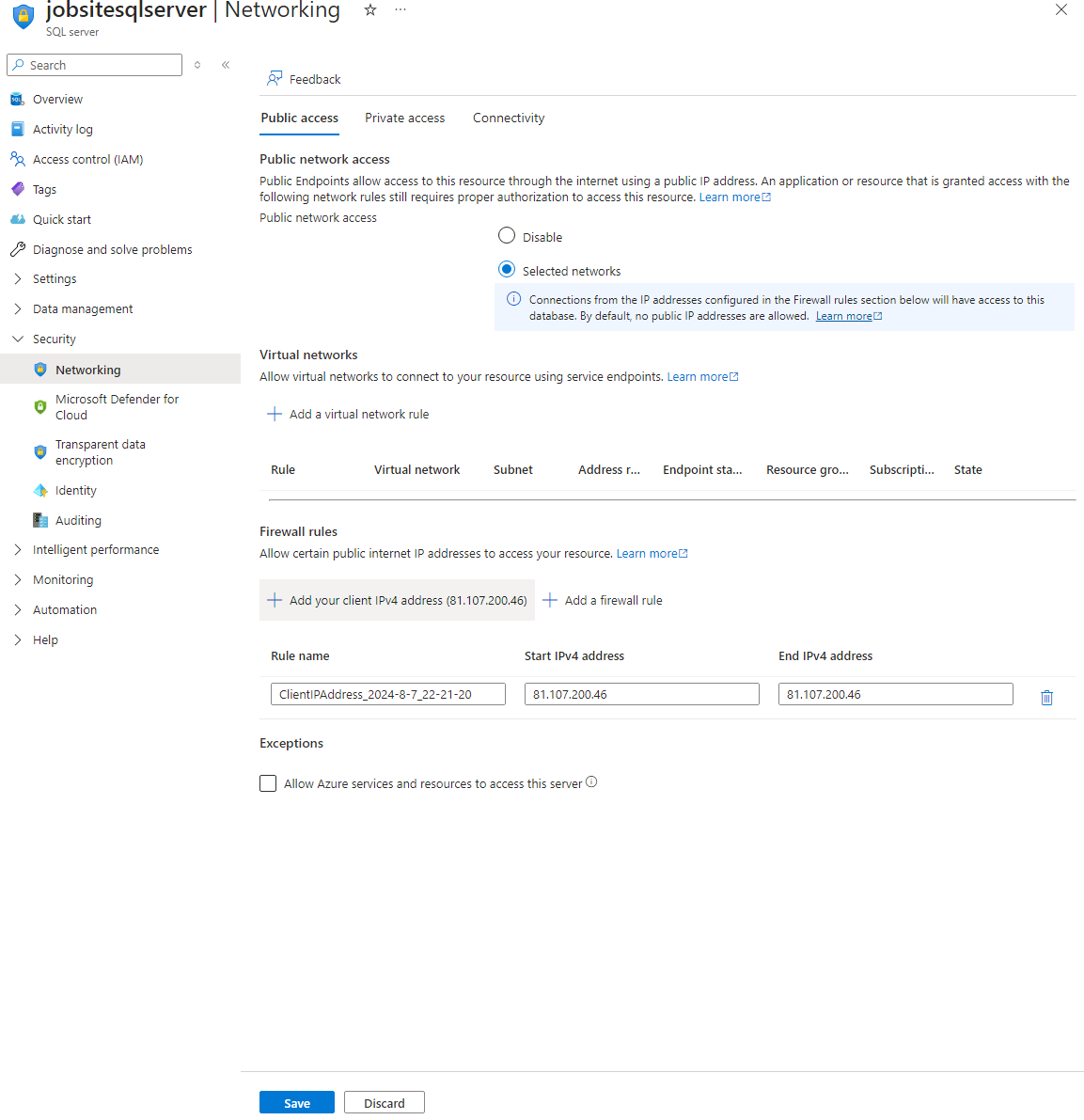
Use SQL Server Management Studio (SSMS) to connect to your Azure SQL Server and create necessary tables. You can use the following instructional video to set up your SSMS and data studio:

https://www.youtube.com/watch?v=r\_StkzGWXiI

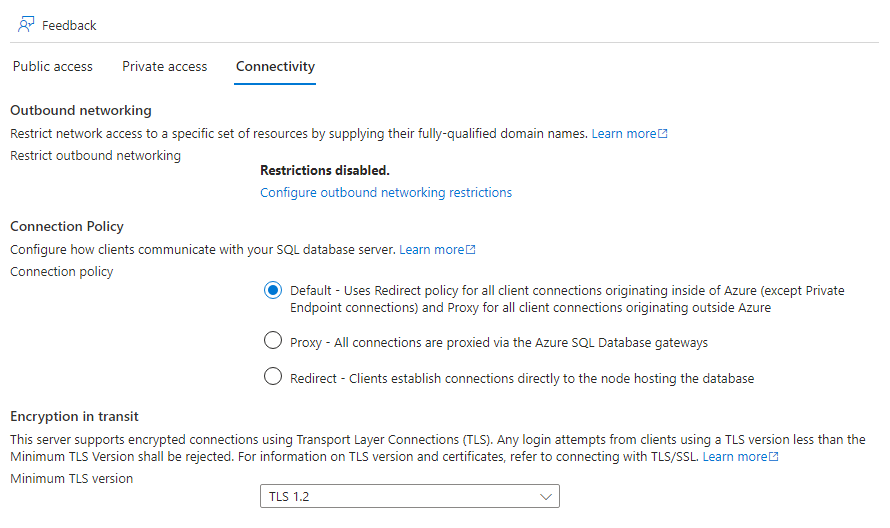
**a. Create Tables**

Connect to your SQL Server:

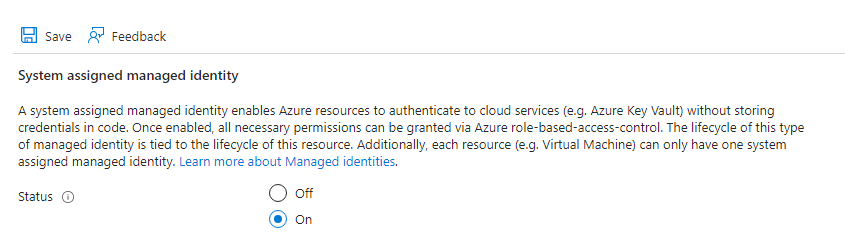
First set the networking settings to allow your ipv4 address in azure in your sql server (click save):



Change the connectivity settings to make sure it is set to default and minimum TLS:



Change the Identity settings as follows :



Click on sql databases, then on the one there , jobsite-db then run the following SQL script to create the necessary tables in Query editor:

```

sql

Copy code

CREATE TABLE Users (

UserID INT PRIMARY KEY IDENTITY(1,1),

Username NVARCHAR(50) NOT NULL,

PasswordHash NVARCHAR(255) NOT NULL,

Qualifications NVARCHAR(255),

Demographics NVARCHAR(255),

Statement NVARCHAR(MAX)

);

CREATE TABLE Applications (

ApplicationID INT PRIMARY KEY IDENTITY(1,1),

UserID INT,

JobTitle NVARCHAR(255),

Status NVARCHAR(50),

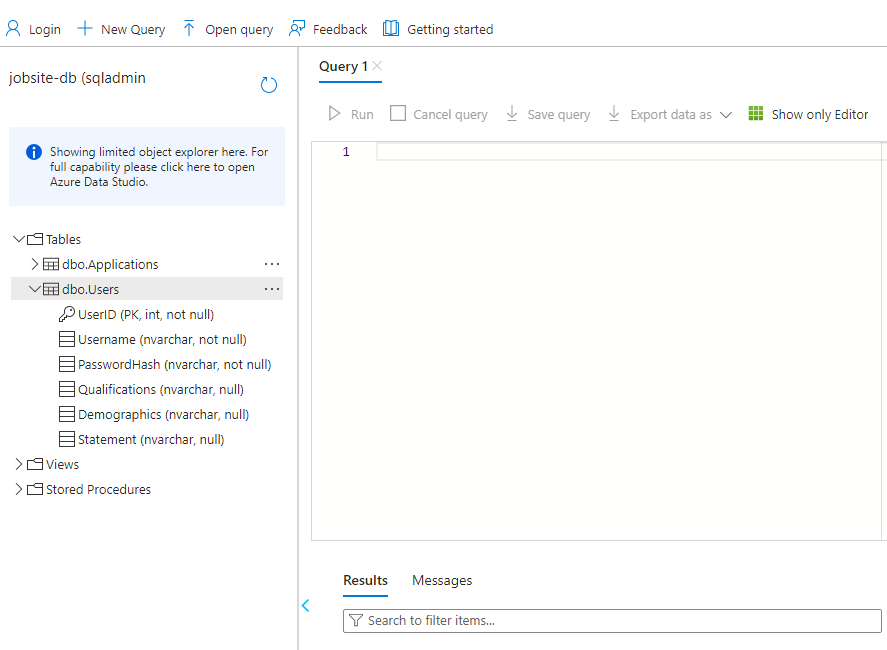
History NVARCHAR(MAX),

FOREIGN KEY (UserID) REFERENCES Users(UserID)

);

```

Your tables should now be visible in the query editor once you refresh

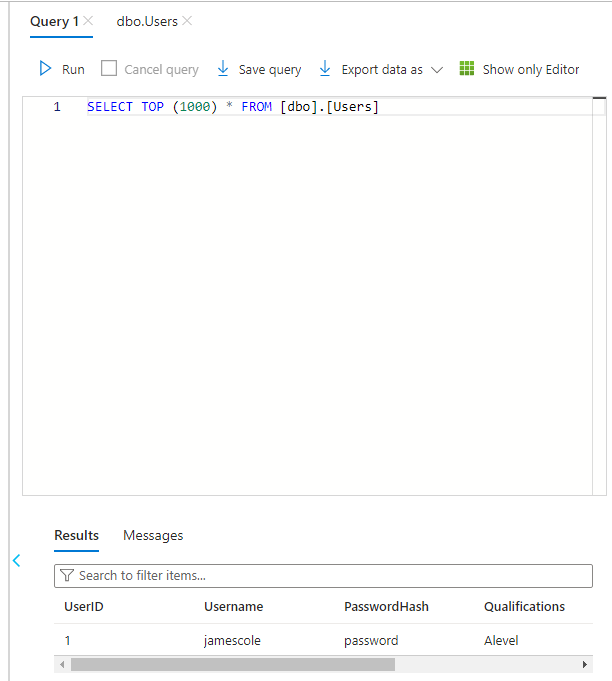


Add a record and then make another query to see this:

```

SELECT TOP (1000) \* FROM [dbo].[Users]

```

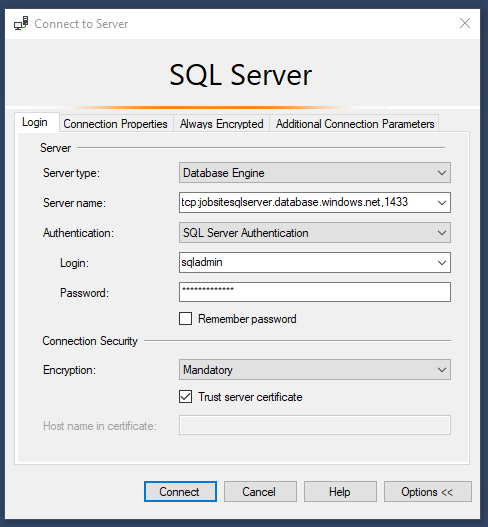


**2.1 Setting Up SSMS (Microsoft sql server management studio)**

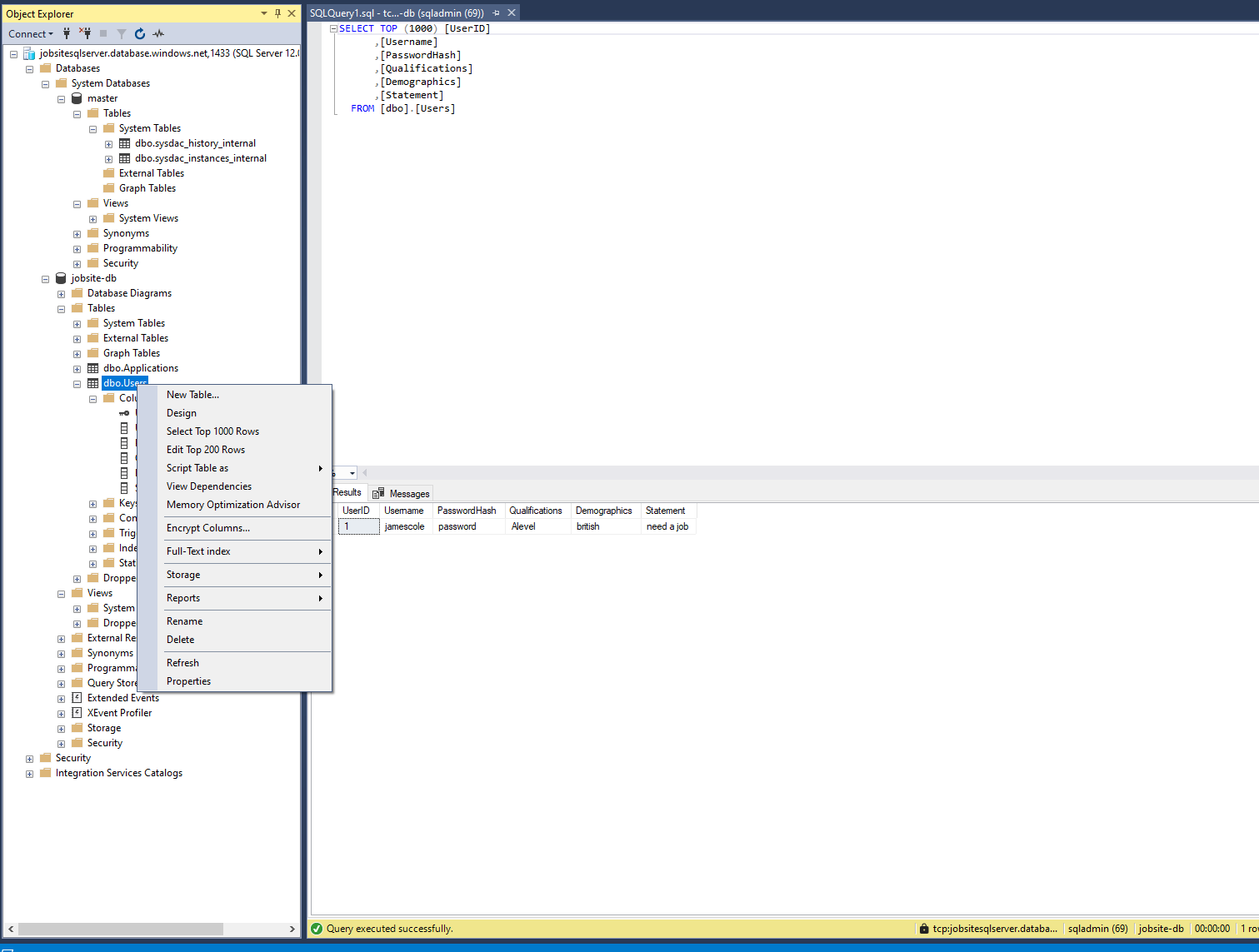
Firstly download SSMS on your machine with the relevant version:

https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

Sign in to sql server using the connection string from your sql database:



Now you can see your data by using queries within ssms:

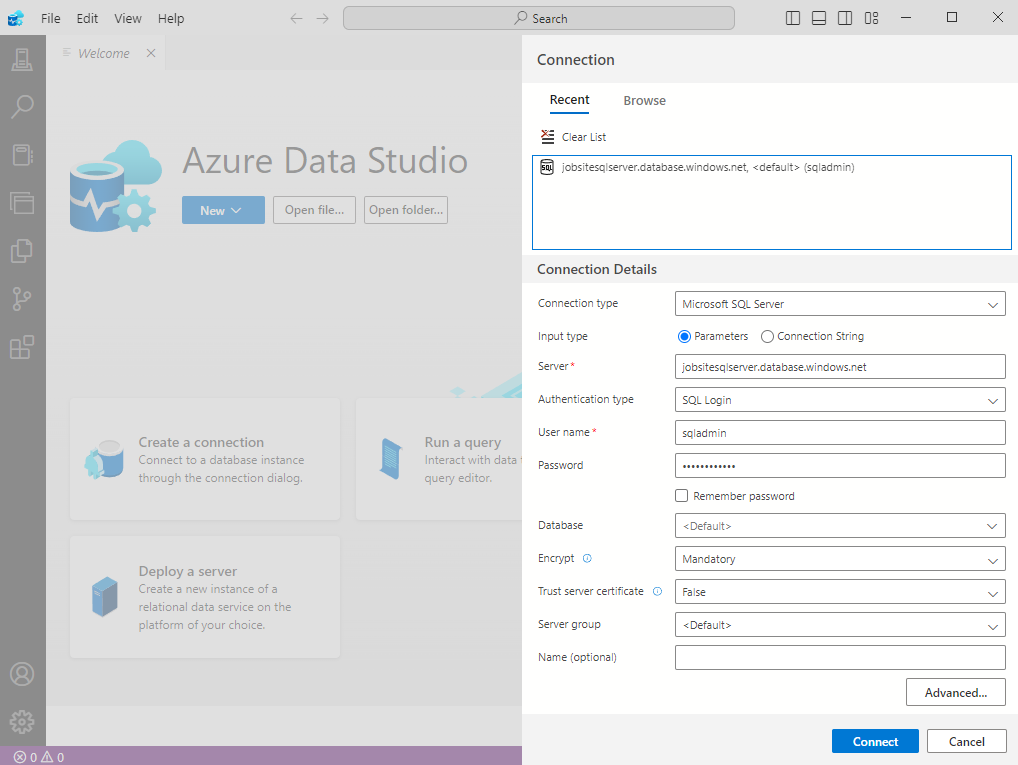


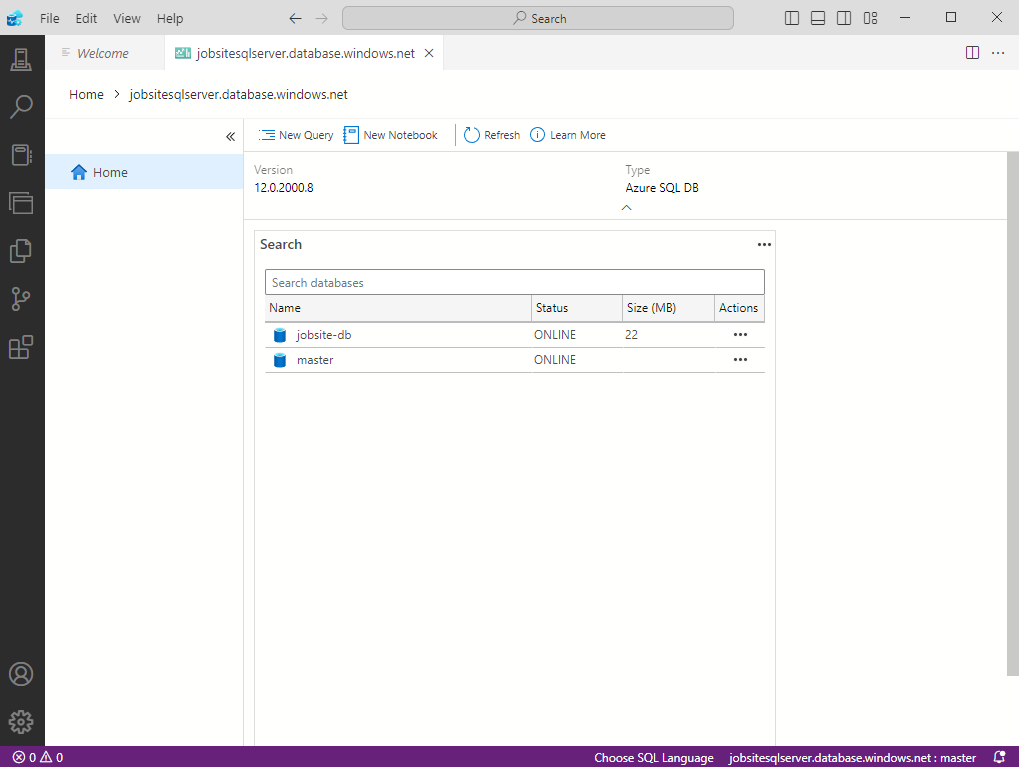
**2.2 Setting Up Azure Data Studio**

Install Azure Data Studio on your machine here :

<https://learn.microsoft.com/en-us/azure-data-studio/download-azure-data-studio?tabs=win-install%2Cwin-user-install%2Credhat-install%2Cwindows-uninstall%2Credhat-uninstall>

firstly set up the connection in Azure Datastudio using your server name and sql login details:



You should now be able to manipulate your data:

**3. Version Control with Azure Repos and Git**

**a. Set Up Azure Repos**

1. Create a new repository in Azure Repos.
2. Clone the repository to your local machine:

Sh

Use token ghp\_X6Y7fDxqcskLGoUl9xztuQQrmp3T582D86wD

Copy code

git clone https://dev.azure.com/your-organization/your-project/\_git/your-repo

1. Add your backend and frontend code to this repository.

**4. Backend Development with Node.js/Express**

**a. Set Up a Node.js/Express Application**

1. Initialize a Node.js project in your repository:

Sh

```

mkdir backend

cd backend

npm init -y

```

1. Install necessary packages:

Sh

```

npm install express body-parser mssql jsonwebtoken bcryptjs

```

1. Create your server file (server.js):

Js

```

const express = require('express');

const bodyParser = require('body-parser');

const sql = require('mssql');

const jwt = require('jsonwebtoken');

const bcrypt = require('bcryptjs');

const app = express();

app.use(bodyParser.json());

// SQL Server configuration

const sqlConfig = {

user: 'sqladmin',

password: 'Password123!',

database: 'jobsite-db',

server: 'jobsitesqlserver.database.windows.net',

pool: {

max: 10,

min: 0,

idleTimeoutMillis: 30000

},

options: {

encrypt: true, // Use this if you're on Windows Azure

trustServerCertificate: false // Change to true for local development

}

};

// Connect to SQL Server

sql.connect(sqlConfig, err => {

if (err) console.error('SQL connection error:', err);

});

// Routes

app.post('/signup', async (req, res) => {

const { username, password, qualifications, demographics, statement } = req.body;

const hashedPassword = await bcrypt.hash(password, 8);

const query = `INSERT INTO Users (Username, PasswordHash, Qualifications, Demographics, Statement)

VALUES ('${username}', '${hashedPassword}', '${qualifications}', '${demographics}', '${statement}')`;

sql.query(query, (err, result) => {

if (err) return res.status(500).send({ message: 'Database error' });

res.send({ message: 'User registered successfully' });

});

});

app.post('/login', async (req, res) => {

const { username, password } = req.body;

const query = `SELECT \* FROM Users WHERE Username = '${username}'`;

sql.query(query, async (err, result) => {

if (err) return res.status(500).send({ message: 'Database error' });

const user = result.recordset[0];

if (!user) return res.status(404).send({ message: 'User not found' });

const isMatch = await bcrypt.compare(password, user.PasswordHash);

if (!isMatch) return res.status(400).send({ message: 'Invalid credentials' });

const token = jwt.sign({ id: user.UserID }, 'yourSecretKey');

res.send({ token });

});

});

app.get('/applications', (req, res) => {

const token = req.headers.authorization.split(' ')[1];

const decoded = jwt.verify(token, 'yourSecretKey');

const query = `SELECT \* FROM Applications WHERE UserID = '${decoded.id}'`;

sql.query(query, (err, result) => {

if (err) return res.status(500).send({ message: 'Database error' });

res.send(result.recordset);

});

});

app.listen(3000, () => console.log('Server running on port 3000'));

```

5.0 Set up on Github with azure app

Connect your app with github and use github as source

