CSE3501

INFORMATION SECURITY ANALYSIS AND AUDIT

DIGITAL ASSIGNMENT(LAB) - 5

LAB SLOT: L31+32

NAME: Abhinaba Mitra
REG. NO.: 18BIT0253

FACULTY: Prof. SUMAIYA THASEEN

TOPIC: SIGNUP NEW USER

Signup New User

Design a form for new user signup which accepts username (register number), password (first name) and email (give valid email of yours which is active) as form fields in the client side. All details are sent to server after submit button is clicked. In the server side, perform encryption/hashing of the password using any built-in functions supported in the environment.

All the user details including the encrypted password are stored in the database. Once the details are stored in the database, a success message is displayed in the form "A new login is created for you (display your register number))"

Complete code and Snapshots of the following to be included

1. Form design - 3 marks

2. Encryption of password – 3 marks

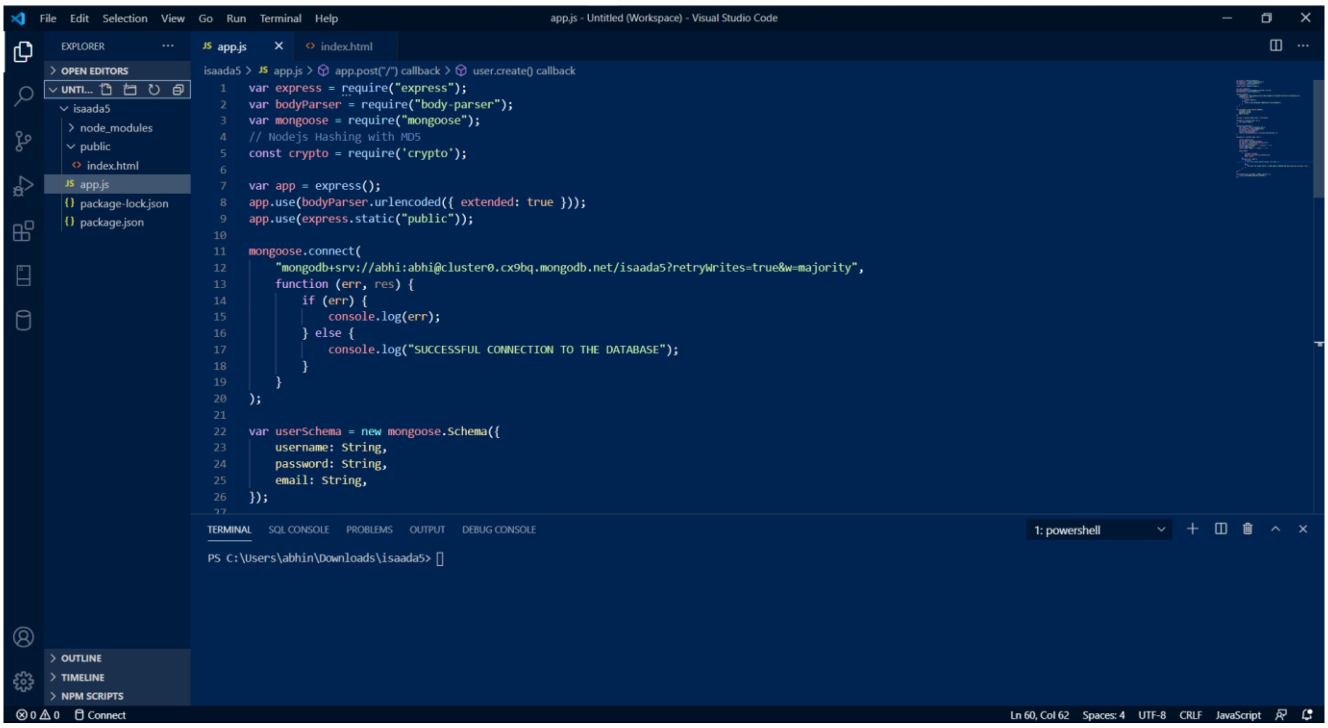
3. Storing in database – 3 marks

4. Success message in client screen – 1 mark

Note:

- *You can use any client-server environment installed on your laptop and run on the browser or GUI of the software.*
- *Write the reason why the specific client-server environment and the encryption/hashing algorithm is chosen for this question.*
- *Only encryption/hashing is performed, decryption not required.*
- *Assignment shared means zero for both parties.*
- *Code can be obtained from internet, if so, provide the link of the same*.

Screenshot of the Visual Studio Code platform with powershell terminal, on which the technology stack is implemented for the problem:


File Edit Selection View Go Run Terminal Help
app.js - Untitled (Workspace) - Visual Studio Code
EXPLORER
OPEN EDITORS
UNTI...
isaada5
node_modules
public
index.html
app.js
package-lock.json
package.json
app.js
index.html
isaada5 > app.js > app.post("/") callback > user.create() callback
TERMINAL SQL CONSOLE PROBLEMS OUTPUT DEBUG CONSOLE
1: powershell
OUTLINE
TIMELINE
NPM SCRIPTS
Connect
Ln 60, Col 62 Spaces: 4 UTF-8 CRLF JavaScript

```javascript
var express = require("express");
var bodyParser = require("body-parser");
var mongoose = require("mongoose");
// Nodejs Hashing with MD5
const crypto = require('crypto');

var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static("public"));

mongoose.connect(
    "mongodb+srv://abhi:abhi@cluster0.cx9bq.mongodb.net/isaada5?retryWrites=true&w=majority",
    function (err, res) {
        if (err) {
            console.log(err);
        } else {
            console.log("SUCCESSFUL CONNECTION TO THE DATABASE");
        }
    }
);

var userSchema = new mongoose.Schema({
    username: String,
    password: String,
    email: String,
});
```

```
PS C:\Users\abhin\Downloads\isaada5> 
```

The Technology Stack/Client-Server Architecture used for the problem is:

FRONTEND (CLIENT SIDE):

HTML, A standard framework for every web application, assisted by CSS and Bootstrap, provides us the interface for the same.
CSS, Cascading Style Sheets used to format, style and decorate the layout of the web pages.

BACKEND (SERVER SIDE):

NodeJS, is a Runtime Environment Server side language for executing JS code, connecting to the database.
ExpressJS, is a flexible NodeJS web application framework that provides a feature for web application APIs.
Body-Parser, a middleware for NodeJS, which parses incoming request bodies in a middleware before the handlers we have used like crypto, under the req.body property.
Crypto, a module/dependency which is a wrapper for OpenSSL cryptographic functions. It supports calculating hashes, authentication with HMAC, Ciphers, etc.
MongoDB Atlas, is the global cloud database service for modern web applications. MongoDB is a no SQL database framework, and Atlas provides the interface for the managing of the database for the website, deployed across AWS, Azure or GCP depending on our choice.

The reason for choosing such specific Client-Server Architecture was because of the current trend of not using legacy technologies like php or .NET in their web app development, these versions of the frontend and backend languages are the tech stack that is used by the companies. Most of the Full Stack Developers use NodeJS and MongoDB for the web app. Moreover, MongoDB Atlas is used in the assignment which is the cloud-based interface deployed on any cloud services of our choice like AWS, Azure or GCP. I have used AWS Services for hosting the MongoDB Collection and connected with the help of Mongoose schema for modelling our Application Data.

Dependencies installed are listed in **package.json** file:

```
{
  "name": "form_authentication",
  "version": "1.0.0",
  "description": "This is done in correspondance to the Nasscom Digital Assignment fo
r Form Authentcation",
  "main": "app.js",
  "scripts": {
    "test": "run app"
  },
  "author": "troj4n",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "crypto": "^1.0.1",
    "express": "^4.17.1",
    "mongoose": "^5.10.11"
  }
}
```

CLIENT SIDE

Sign-Up Page Code:

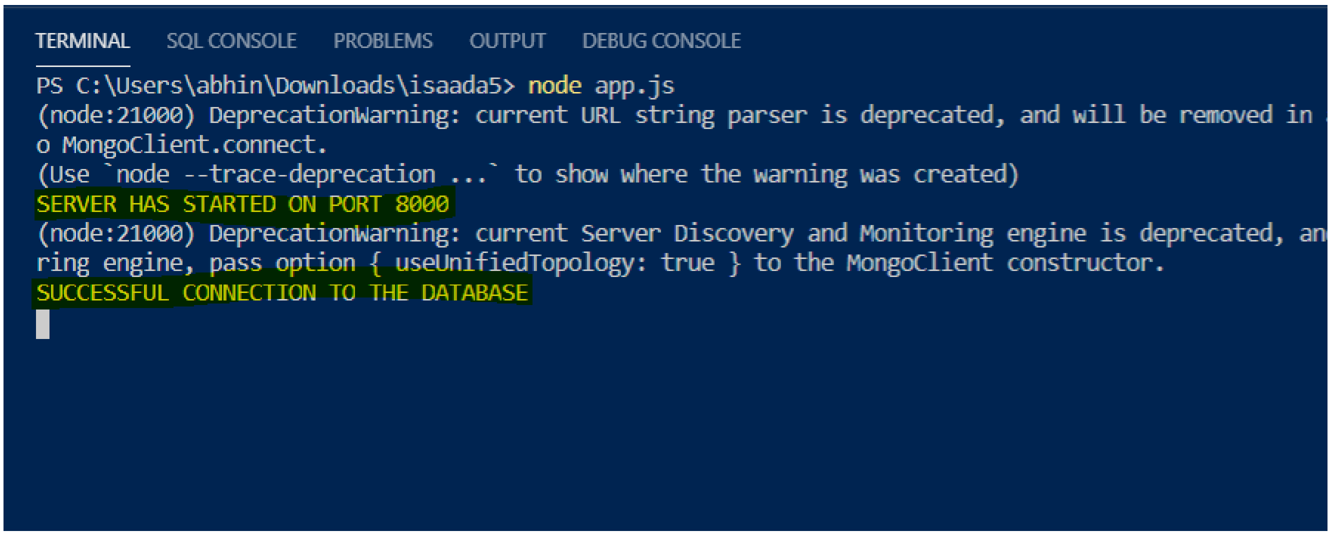
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Form Authentication</title>
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.4.1/css/
all.css" crossorigin="anonymous">
        <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700" r
el="stylesheet">
        <style>
            html,
            body {
                min-height: 100%;
            }

            body,
            div,
            form,
            input,
            select,
            p {
                padding: 0;
                margin: 0;
                outline: none;
                font-family: Roboto, Arial, sans-serif;
                font-size: 16px;
                color: #eee;
            }

            body {
                background: url("https://www.w3docs.com/uploads/media/default/0001/01
/b5edc1bad4dc8c20291c8394527cb2c5b43ee13c.jpeg") no-repeat center;
                background-size: cover;
            }

            h1,
            h2 {
                text-transform: uppercase;
                font-weight: 400;
            }

            h2 {
                margin: 0 0 0 8px;
            }

            .main-block {
                display: flex;
                flex-direction: column;
                justify-content: center;
                align-items: center;
                height: 100%;
                padding: 25px;
                background: rgba(0, 0, 0, 0.5);
            }

            .left-part,
            form {
                padding: 25px;
            }

            .left-part {
                text-align: center;
            }

            .fa-graduation-cap {
                font-size: 72px;
            }

            form {
                background: rgba(0, 0, 0, 0.7);
            }

            .title {
                display: flex;
                align-items: center;
                margin-bottom: 20px;
            }

            .info {
                display: flex;
```

```css
    flex-direction: column;
}

input,
select {
    padding: 5px;
    margin-bottom: 30px;
    background: transparent;
    border: none;
    border-bottom: 1px solid #eee;
}

input::placeholder {
    color: #eee;
}

option:focus {
    border: none;
}

option {
    background: black;
    border: none;
}

.checkbox input {
    margin: 0 10px 0 0;
    vertical-align: middle;
}

.checkbox a {
    color: #26a9e0;
}

.checkbox a:hover {
    color: #85d6de;
}

.btn-item,
button {
    padding: 10px 5px;
    margin-top: 20px;
    border-radius: 5px;
    border: none;
    background: #26a9e0;
    text-decoration: none;
    font-size: 15px;
    font-weight: 400;
    color: #fff;
}

.btn-item {
    display: inline-block;
```

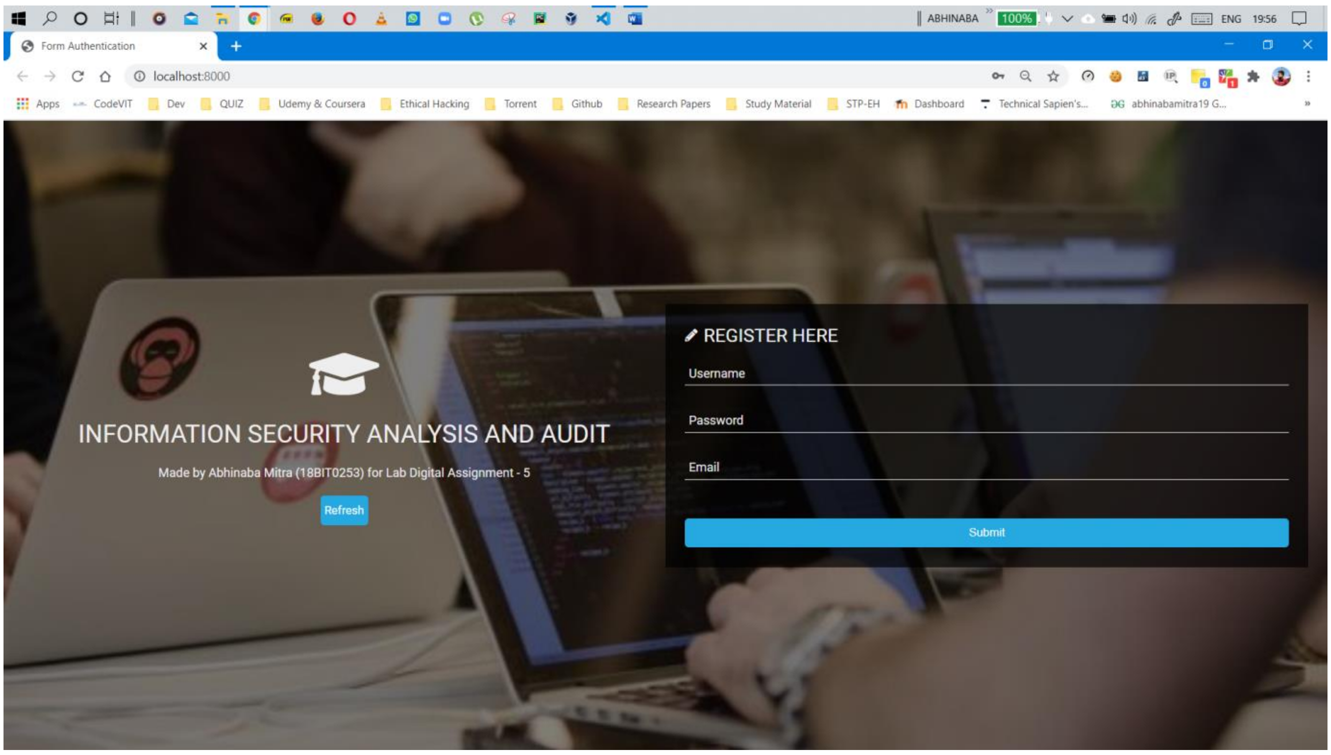
```
                margin: 20px 5px 0;
            }

            button {
                width: 100%;
            }

            button:hover,
            .btn-item:hover {
                background: #85d6de;
            }

            @media (min-width: 568px) {

                html,
                body {
                    height: 100%;
                }

                .main-block {
                    flex-direction: row;
                    height: calc(100% - 50px);
                }

                .left-part,
                form {
                    flex: 1;
                    height: auto;
                }
            }
        </style>
    </head>
    <body>
        <div class="main-block">
        <div class="left-part">
            <i class="fas fa-graduation-cap"></i>
            <h1>Information Security Analysis and Audit</h1>
            <p>Made by Abhinaba Mitra (18BIT0253) for Lab Digital Assignment - 5
            </p>
            <div class="btn-group">
                <a HREF="javascript:history.go(0)" class="btn-item">Refresh</a>
            </div>
        </div>
        <form action="/" method = "POST">
            <div class="title">
                <i class="fas fa-pencil-alt"></i>
                <h2>Register here</h2>
            </div>
            <div class="info">
                <input type="text" id="username" name="username" placeholder="Usernam
e">
                <input type="password" id="password" name="password" placeholder="Pas
sword">
```

```
                <input type="text" id="email" name="email" placeholder="Email">
            </div>
            <button type="submit" value="submit">Submit</button>
        </form>
        </div>
        </div>
    </body>
</html>
```

The code under <style> tag of HTML consists of the CSS section. It consists of a form which takes input username as the Registration Number, Email ID of the user and password as the first name of the user. So, our server side code is written in app.js file using ExpressJS and NodeJS which is now run using the Powershell in VS Code on localhost on port 8000. It will show the success message on successful connection.

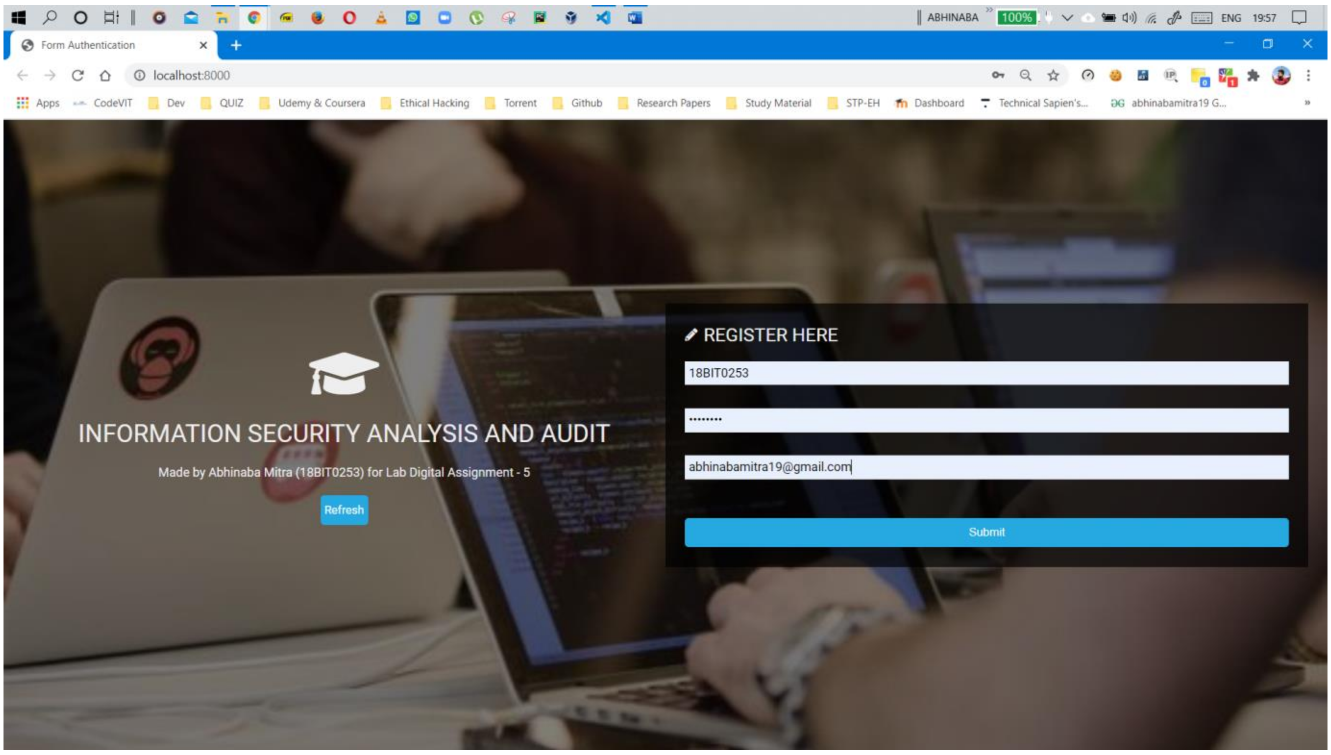


```
PS C:\Users\abhin\Downloads\isaada5> node app.js
(node:21000) DeprecationWarning: current URL string parser is deprecated, and will be removed in
o MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
SERVER HAS STARTED ON PORT 8000
(node:21000) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, an
ring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
SUCCESSFUL CONNECTION TO THE DATABASE
```


Form Authentication
localhost:8000
REGISTER HERE
Username
Password
Email
Submit
INFORMATION SECURITY ANALYSIS AND AUDIT
Made by Abhinaba Mitra (18BIT0253) for Lab Digital Assignment - 5
Refresh

Form Design

We then fill up the details with my registration number as the username, valid email id as the id and my name as the password.

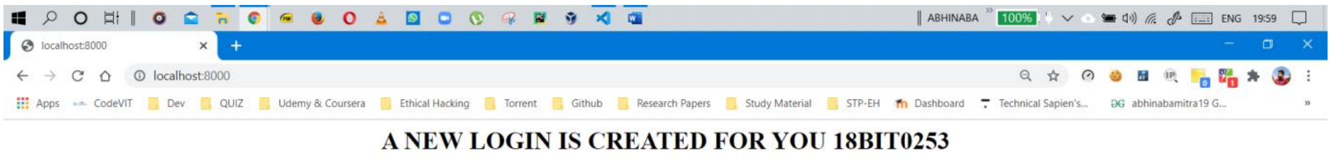

Form Authentication
localhost:8000
INFORMATION SECURITY ANALYSIS AND AUDIT
Made by Abhinaba Mitra (18BIT0253) for Lab Digital Assignment - 5
Refresh
REGISTER HERE
Submit

18BIT0253

••••••••

abhinabamitra19@gmail.com

After clicking on Submit we get, the following **success message**,



A NEW LOGIN IS CREATED FOR YOU 18BIT0253

Success Message in Client screen

SERVER SIDE

I have used MongoDB Atlas Tool to see the interface of the data stored where password is stored in encrypted form, and also shown in the the node powershell window od VS Code.

SERVER SIDE CODE WITH EXPLANATION

```
// The dependencies are loaded onto a variable
```

```
var express = require("express");
var bodyParser = require("body-parser");
var mongoose = require("mongoose");
// Nodejs Hashing with MD5
const crypto = require('crypto');

// We create an instance of express, which is the used to parse the other dependencie
s

var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static("public"));

// We use the mongoose schema to decorate the application data or to parse the data.
// The link here is the link which connects the online hosted MongoDB database to the
 localhost.
// As mentioned in the link, isaada5 is the name of the database stored.

mongoose.connect(
    "mongodb+srv://abhi:abhi@cluster0.cx9bq.mongodb.net/isaada5?retryWrites=true&w=ma
jority",
    function (err, res) {
        if (err) {
            console.log(err);
        } else {
            console.log("SUCCESSFUL CONNECTION TO THE DATABASE");
        }
    }
);

// This defines the model of the schema and then connects with the help of Mongoose a
nd stores it under the name user.

var userSchema = new mongoose.Schema({
    username: String,
    password: String,
    email: String,
});

var user = mongoose.model("user", userSchema);

// The app.get function renders the index.html file (client-side web page)

app.get("/", function (req, res) {
    res.render("index");
})

// This is the main function where it encypts the password into MD5 hashes, and store
s in the database online

function encrypt(text) {
    let cipher = crypto.createHash('md5');
    let encrypted = cipher.update(text);
    encrypted = cipher.digest();
    console.log(encrypted);
    return { encryptedPassword: encrypted.toString('hex') };
}

// the app.post function where it pulls all the thread together, and finishing connec
tion from client to server

app.post("/", function (req, res) {

    console.log(req.body)
    var username = req.body.username;
    var password = encrypt(req.body.password);
    var email = req.body.email;
    console.log("{ username: '" + username + "' }")
    console.log(password)
    console.log("{ email: '" + email + "' }")

    user.create(
        {
            username: username,
            password: password.encryptedPassword,
            email: email,
        },
        function (err, data) {
            if (err) {
                res.send("SERVER ERROR OCCURRED, TRY AGAIN.")
            }
            else {
                res.send(`<h1 align="center"> A NEW LOGIN IS CREATED FOR YOU ${req.bo
dy.username} <\h1>`)
            }
        }
    );
});

// the app.listen function which specifies the port on which the server listens to

app.listen(process.env.PORT || 8000, function () {
    console.log("SERVER HAS STARTED ON PORT 8000");
});
```

The hashing technique used for storing of the password in the web application is **MD5**. The code for encryption of the password is there in the encrypt function of the above code.

The respective Hashing Algorithm is chosen because:

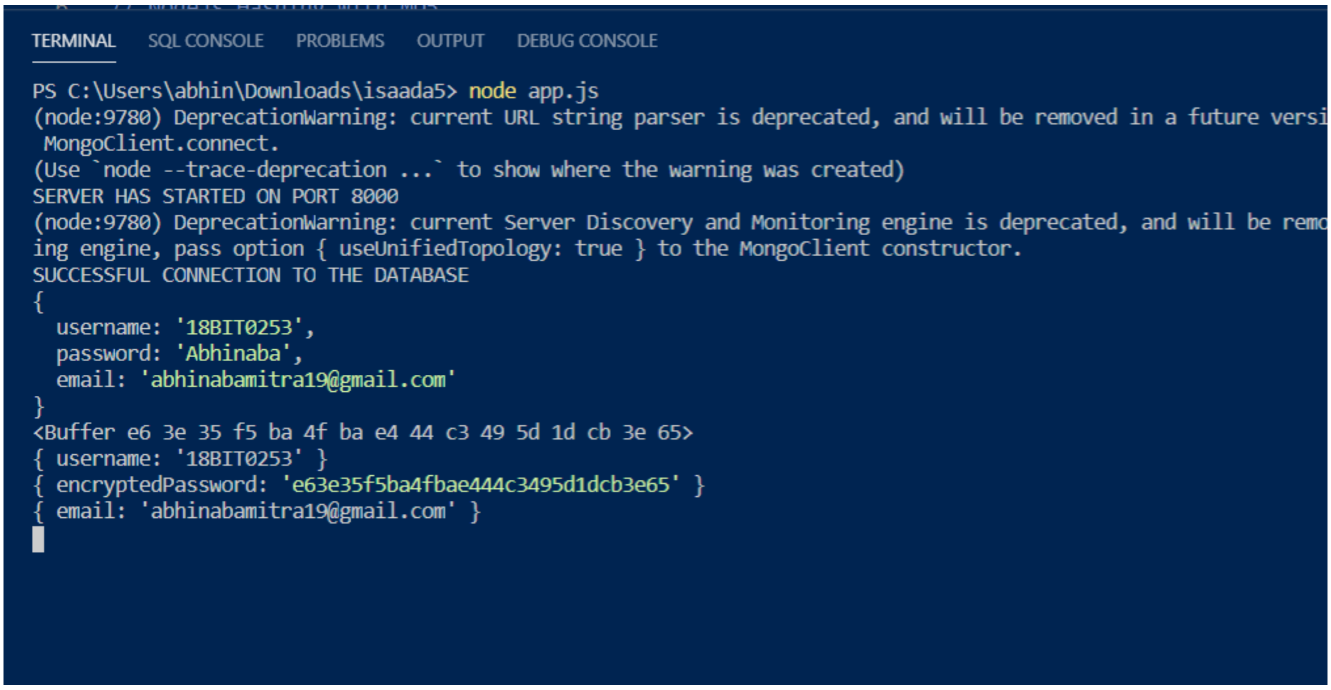
MD5 is the most widely used hash function in the computer domain (which can be called a hashing algorithm, a digest algorithm), which is used to ensure the integrity and consistency of a message, in our case the password.

The MD5 algorithm has the following features:
1. Compressibility: Any length of data, the calculated length of the MD5 value is fixed.
2. Easy to calculate: It is easy to figure out the MD5 value from the original data.
3. Anti-Modification: Make any changes to the original data, even if only one byte is modified, the resulting MD5 value is very different.
4. Strong Anti-Collision: known raw data and its MD5 value, it is very difficult to find a pseudo-data with the same MD5 value.

MD5 's role is to allow bulk information to be compressed into a confidential format before signing a private key with a digital signature software (that is, converting any string of any length into a long hexadecimal number string).

OUTPUT

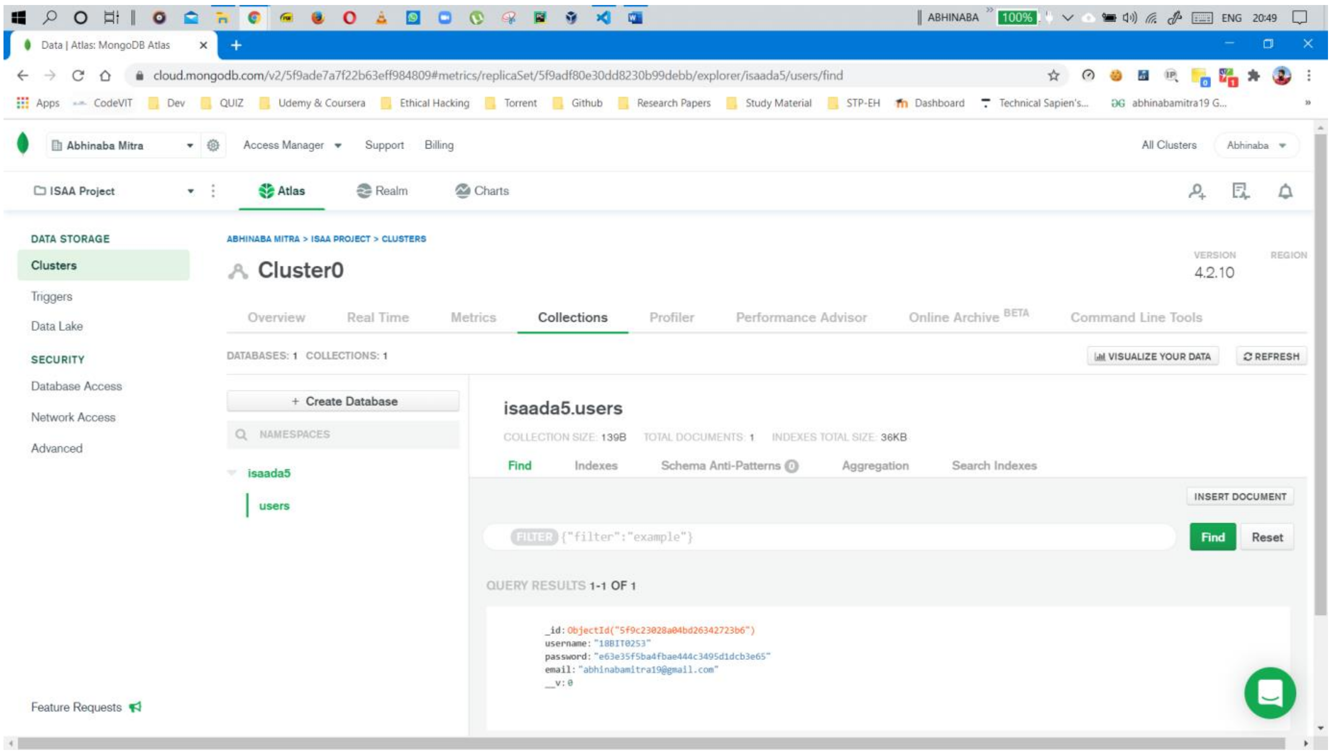
Encryption of Password as shown in the VS Code PowerShell Terminal of NodeJS:
After clicking Submit, we get to see the following window,



```
PS C:\Users\abhin\Downloads\isaada5> node app.js
(node:9780) DeprecationWarning: current URL string parser is deprecated, and will be removed in a future versi
 MongoClient.connect.
(Use `node --trace-deprecation ...` to show where the warning was created)
SERVER HAS STARTED ON PORT 8000
(node:9780) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be remo
ing engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
SUCCESSFUL CONNECTION TO THE DATABASE
{
  username: '18BIT0253',
  password: 'Abhinaba',
  email: 'abhinabamitra19@gmail.com'
}
<Buffer e6 3e 35 f5 ba 4f ba e4 44 c3 49 5d 1d cb 3e 65>
{ username: '18BIT0253' }
{ encryptedPassword: 'e63e35f5ba4fbae444c3495d1dcb3e65' }
{ email: 'abhinabamitra19@gmail.com' }
```

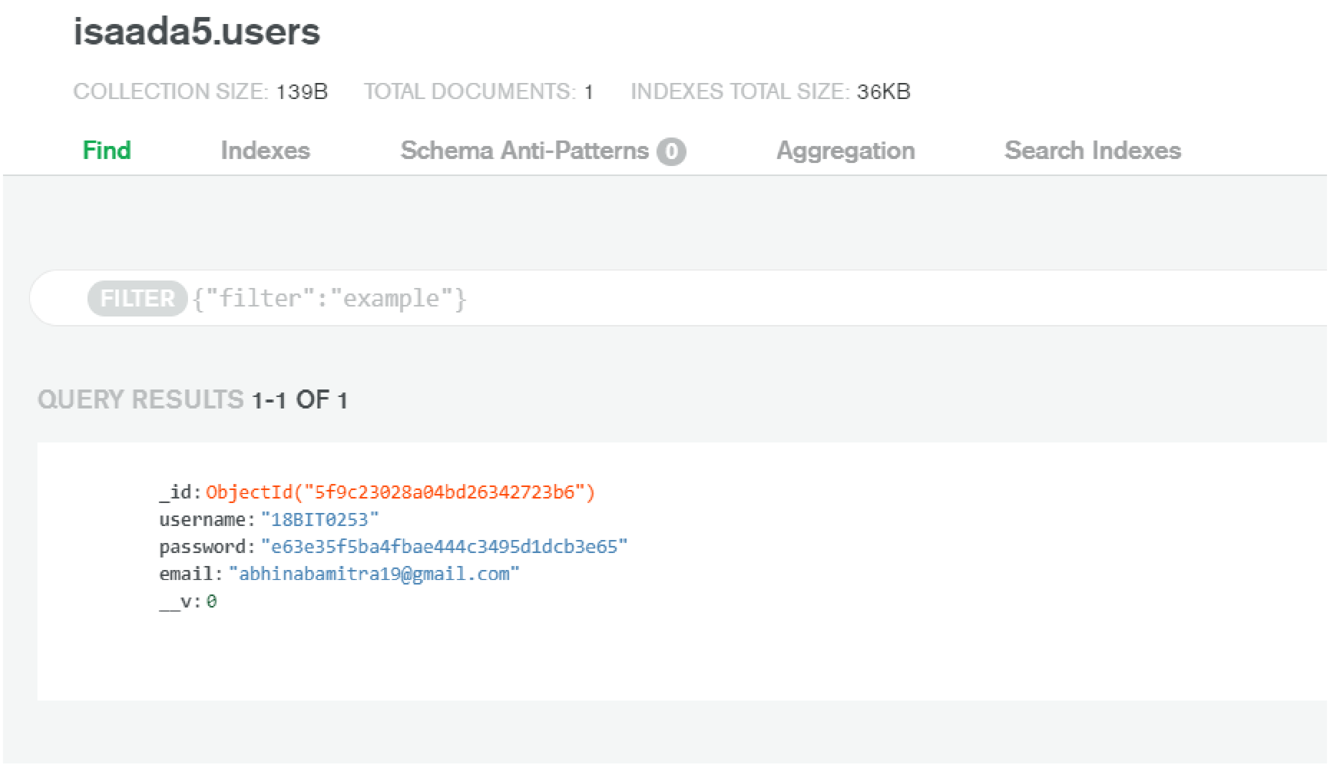
As we see, the original data and the data stored after hashing of password. The Final Buffer is also printed which is then converted to String and stored in the database.

Now **Storing in Database**,


isaada5.users
COLLECTION SIZE: 139B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 36KB
QUERY RESULTS 1-1 OF 1

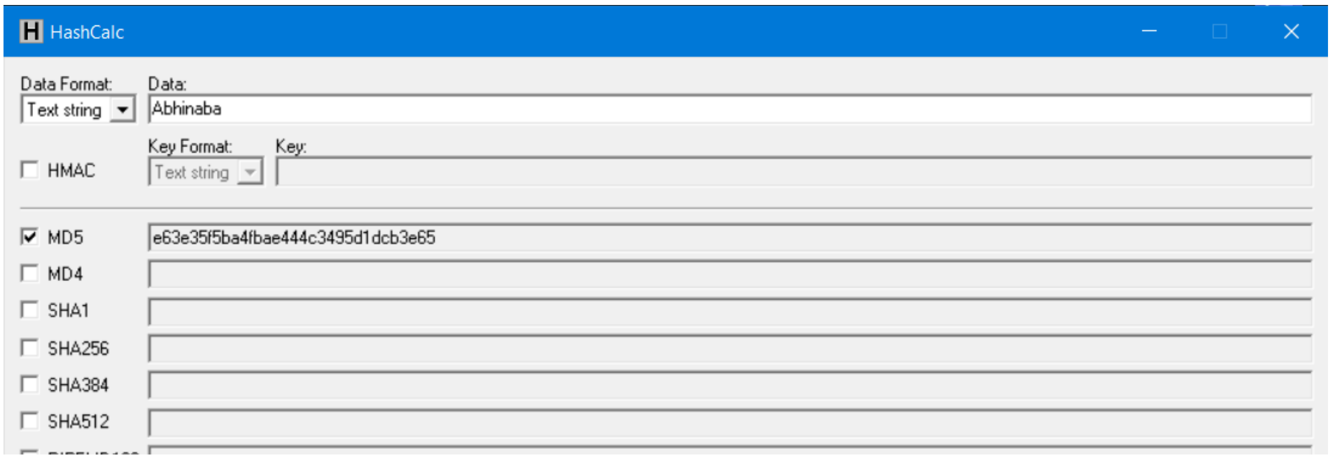
```
_id: ObjectId("5f9c23028a04bd26342723b6")
username: "18BIT0253"
password: "e63e35f5ba4fbae444c3495d1dcb3e65"
email: "abhinabamitra19@gmail.com"
__v: 0
```

As already mentioned the name of our database is **isaada5** and we get to see the **users** schema under that collection and the data stored.



```
_id: ObjectId("5f9c23028a04bd26342723b6")
username: "18BIT0253"
password: "e63e35f5ba4fbae444c3495d1dcb3e65"
email: "abhinabamitra19@gmail.com"
__v: 0
```

We get the Hash of Password ‘Abhinaba’ as e63e35f5ba4fbae444c3495d1dcb3e65. In order to check if the MD5 Hash is coming out to be right, we use HashCalc in order to check the value of the same String.



HashCalc

Data Format: Text string
Data: Abhinaba

☐ HMAC Key Format: Text string Key:

☑ MD5 e63e35f5ba4fbae444c3495d1dcb3e65
☐ MD4
☐ SHA1
☐ SHA256
☐ SHA384
☐ SHA512

As we can see it gives us the same Hash value of e63e35f5ba4fbae444c3495d1dcb3e65 for the string 'Abhinaba'.

REFERENCES

[1] https://www.w3docs.com/learn-html/html-form-templates.html for frontend CSS.
[2] https://topic.alibabacloud.com/a/nodejs-crypto-module-md5-and-hmac-encryption_1_11_30518380.html referred for the encryption function purposes.

--THANK YOU--

Abhinaba Mitra

Digitally signed by Abhinaba Mitra
Date: 2020.10.30 21:15:52 +05'30'