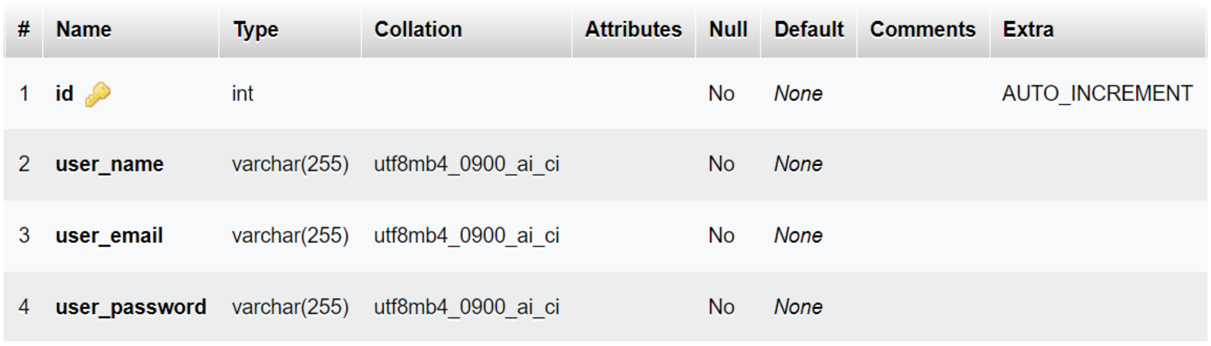
Lab Exercise: User Authentication and Authorization

Overview of the lab exercise:

This lab exercise guides you through the process of implementing user authentication and authorization in a web application. You'll start by setting up a database to store user information. Then, you'll create a login form using Pug, a template engine for Node.js. On the server side, you'll handle form submissions by verifying the submitted credentials against the database records. You may user the previous app project to work on this lab exercise.

Part 1: Table Setup in MySQL

1. Setup a table named 'users' in the existing database. The table structure as follows:



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**id** 🔑	int			No	*None*		AUTO_INCREMENT
2	**user_name**	varchar(255)	utf8mb4_0900_ai_ci		No	*None*		
3	**user_email**	varchar(255)	utf8mb4_0900_ai_ci		No	*None*		
4	**user_password**	varchar(255)	utf8mb4_0900_ai_ci		No	*None*		

Part 2: Create Pug pages

1. Create 2 Pug pages, login.pug and dashboard.pug. The basic contents for both pages as follows:

login.pug

```
doctype html
html
  head
    title Login Page
  body
    h1 Login
    form(action="/login" method="post")
      div
        label(for="useremail") Email:
        input(type="email" id="useremail" name="useremail")
      div
        label(for="userpassword") Password:
        input(type="password" id="userpassword" name="userpassword")
      div
        input(type="submit" value="Login")
```

```
dashboard.pug

doctype html
html
  head
    title Dashboard
  body
    h1 Welcome, #{user_name}!
    a(href="logout") Logout
```

Part 3: `express-session` Package Installation and Setup

1. Install the express-session package using npm

```
npm install express-session
```

2. Require it in your application (in app.js) and set up the middleware

```
const session = require('express-session');

app.use(session({
  secret: 'fwdd',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: false } // Note: the `secure` option should be
enabled only if you are serving your app over HTTPS
}));
```

3. Add the login route to set the session after a successful login

```
app.get('/login', (req, res) => {
  res.render('login');
});

// Handle login form submission
app.post('/login', (req, res) => {
  let sql = 'SELECT * FROM users WHERE user_email = ? AND user_password =
?';
  let query = db.query(sql, [req.body.useremail, req.body.userpassword],
(err, result) => {
    if (err) throw err;
    if (result.length > 0) {
      // Login successful, set session and redirect to dashboard
      req.session.user = result[0]; // Save the user object to the
session
      req.session.user_name = result[0].user_name;
      res.redirect('/dashboard');
    } else {
      // Login failed, respond with error message
      res.send('Login failed');
    }
  });
});
```

4. Add the dashboard route and check for the existence of req.session.user in the routes to control access based on whether the user is logged in or not.

```
app.get('/dashboard', (req, res) => {
  if (!req.session.user) {
    // User is not logged in, redirect to login page
    res.redirect('/login');
  } else {
    // User is logged in, render the dashboard
    res.render('dashboard', { user_name: req.session.user_name });

  }
});
```

5. Clear the session when the user logs out.

```
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if(err) {
      // Handle error
      console.log(err);
      res.send('Error occurred during logout');
    } else {
      // Redirect to login page after successful logout
      res.redirect('/login');
    }
  });
});
```