C237: Software Application Development

LESSON 11A – FORM VALIDATION AND AUTHENTICATION

9.15AM – 11.30AM

Form Validation

Form Validation

- Form validation ensures the correctness and completeness of data entered by the user.
- Ensures data integrity and user experience.
- Types of validation:
  - **Client-Side Validation:** Performed in the browser using JavaScript. Provides immediate feedback but is not secure.
  - **Server-Side Validation:** Performed on the server. Ensures data integrity even if client-side validation is bypassed.
  - **Example:** Validating email and password fields to ensure they meet certain criteria (e.g., email format, password length).

Client-Side Validation

- Done in the browser
- Can be bypassed by malicious users
- Provides immediate feedback to users without requiring a server round-trip.
- Implemented using HTML5 attributes (e.g., required, minlength, maxlength, min) and JavaScript for additional checks.

```
<div class="form-group">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" class="form-control" required>
</div>
<div class="form-group">
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" class="form-control" required>
</div>
```

Client-side Validation – Javascript (Example)

```
</head>
<script>
    function validateForm() {
        // Get form fields
        var password = document.getElementById('password').value;

        // Get error display elements
        var passwordError = document.getElementById('passwordError');

        // Reset errors
        passwordError.innerHTML = '';

        var isValid = true;

        // Password validation
        if (password === '') {
            passwordError.innerHTML = 'Password is required';
            isValid = false;
        } else if (password.length < 6) {
            passwordError.innerHTML = 'Password must be at least 6 characters';
            isValid = false;
        }

        return isValid;
    }
</script>
<body>
```

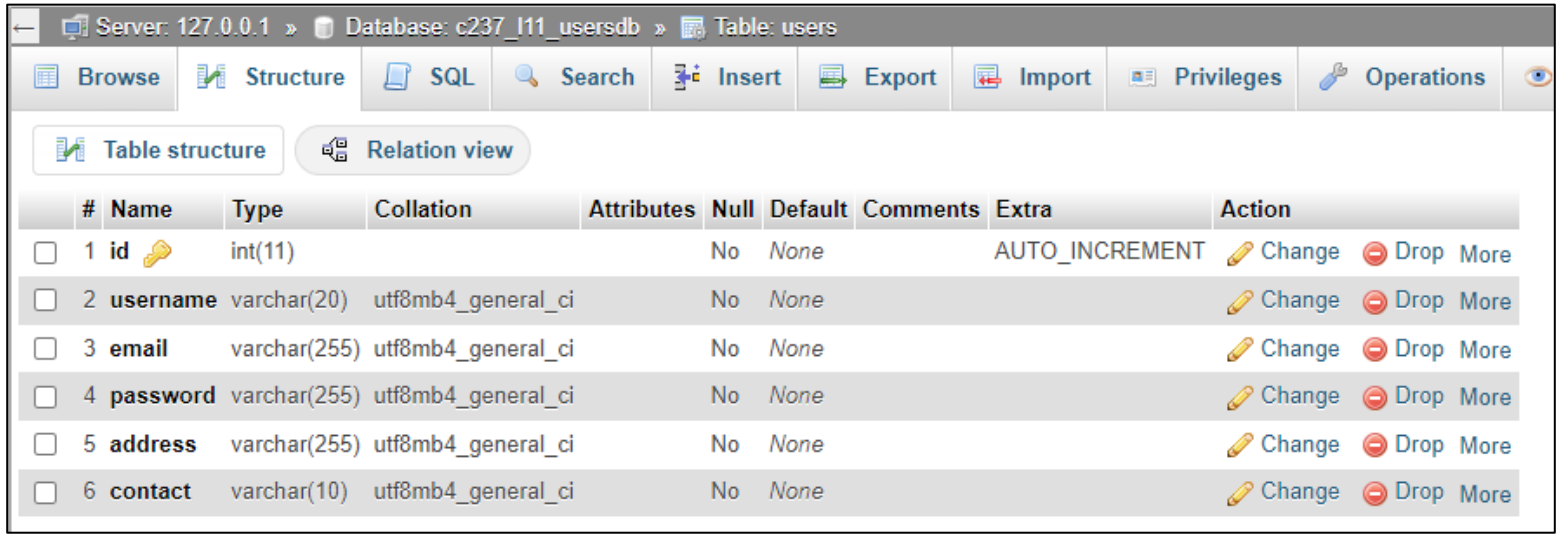
Server-Side Validation (Example)

- Ensures data integrity and security as validation happens on the server.
- Cannot be bypassed by the client.
- **Express Middleware:** A function that processes requests before they reach your routes. Useful for validation.

```
const validateRegistration = (req, res, next) => {
    const { username, email, password, address, contact } = req.body;

    if (!username || !email || !password || !address || !contact || !dateOfBirth) {
        return res.status(400).send('All fields are required.');
    }

    if (!email.includes('@') || password.length < 6) {
      return res.status(400).send('Invalid input');
    }
    next();
};

app.post('/register', validateRegistration, (req, res) => {
    //registration logic here
});
```

Let’s Try!

- Create a folder L11 in your C237 folder.
- Copy the folder registrationApp found in your “L11 Activity Files” folder and paste it into the L11 folder which you created earlier.
- Import your database in phpMyAdmin using the “C237_L11_usersdb.sql” file found in your “L11 Activity Files” folder.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	username	varchar(20)	utf8mb4_general_ci		No	None			Change Drop More
3	email	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
4	password	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
5	address	varchar(255)	utf8mb4_general_ci		No	None			Change Drop More
6	contact	varchar(10)	utf8mb4_general_ci		No	None			Change Drop More

Let’s Try!

- Open the registrationApp (the one in your L11 folder) in Visual Studio Code.
- Open terminal and install the following:
  - **npm install connect-flash:**
    - a middleware for Express applications that allows you to store and retrieve temporary messages, which are typically used for notifications or error messages. These messages are stored in the session and are available only for the next request, after which they are removed. This is useful for displaying messages to the user after a redirect or any action that changes the state of the application.
  - **npm install express-session:**
    - used to install the express-session package, which is a middleware for managing sessions in Express applications. Sessions allow you to store data on the server side, which can be used across multiple requests from the same client.

Set up express-session for session management.

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

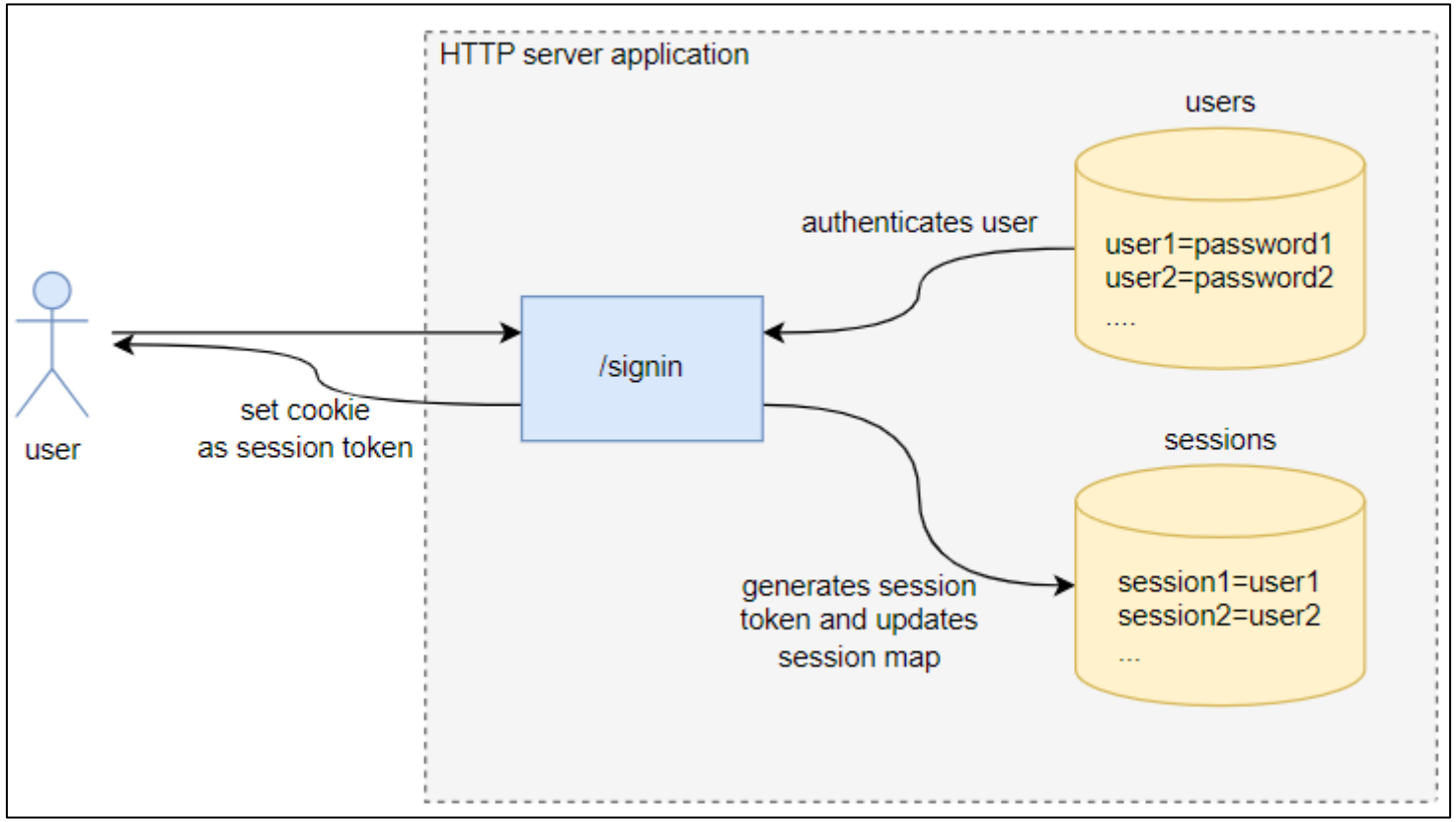
```
const express = require('express');
const mysql = require('mysql2');

//TO DO: Insert code to import 'express-session'
const session = require('express-session');

const flash = require('connect-flash');

const app = express();
```

```
app.use(express.urlencoded({ extended: false }));
app.use(express.static('public'));

//TO DO: Insert code for Session Middleware below
app.use(session({
    secret: 'secret',
    resave: false,
    saveUninitialized: true,
    // Session expires after 1 week of inactivity
    cookie: { maxAge: 1000 * 60 * 60 * 24 * 7 }
}));
```

Session Middleware

- The session middleware in Express.js is used to manage user sessions in a web application.
- Sessions are used to store data about a user's interaction with a web application across multiple requests.
- This is useful for maintaining user state, such as login status, preferences, and other data that should persist throughout the user's visit to the application.
- Examples:
  - Storing the username upon log-in, and displaying it on all other pages

Session Middleware – How it works



https://www.sohamkamani.com/nodejs/session-cookie-authentication/

When a user logs in successfully, a session cookie will be created and sent along with subsequent request.

Implement Server-Side Validation

- Add server-side validation to an existing registration form.
- Steps:
  1. Create a middleware function validateForm.
  2. Integrate validateForm into the registration route.
  3. Test the validation by submitting various forms.

1. Create a middleware function validateRegistration

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

```
//TO DO: Create a middleware function validateRegistration
const validateRegistration = (req, res, next) => {
    const { username, email, password, address, contact } = req.body;

    if (!username || !email || !password || !address || !contact) {
        return res.status(400).send('All fields are required.');
    }

    if (password.length < 6) {
        req.flash('error', 'Password should be at least 6 or more characters long');
        req.flash('formData', req.body);
        return res.redirect('/register');
    }
    next(); //If all validations pass, the next function is called, allowing the request to proceed to the
            //next middleware function or route handler.
};
```

This function checks for the presence of required fields and enforces a minimum password length (6). If any validation fails, it sends an appropriate response or redirects the user with an error message.

2. Integrate validateRegistration into the register route.

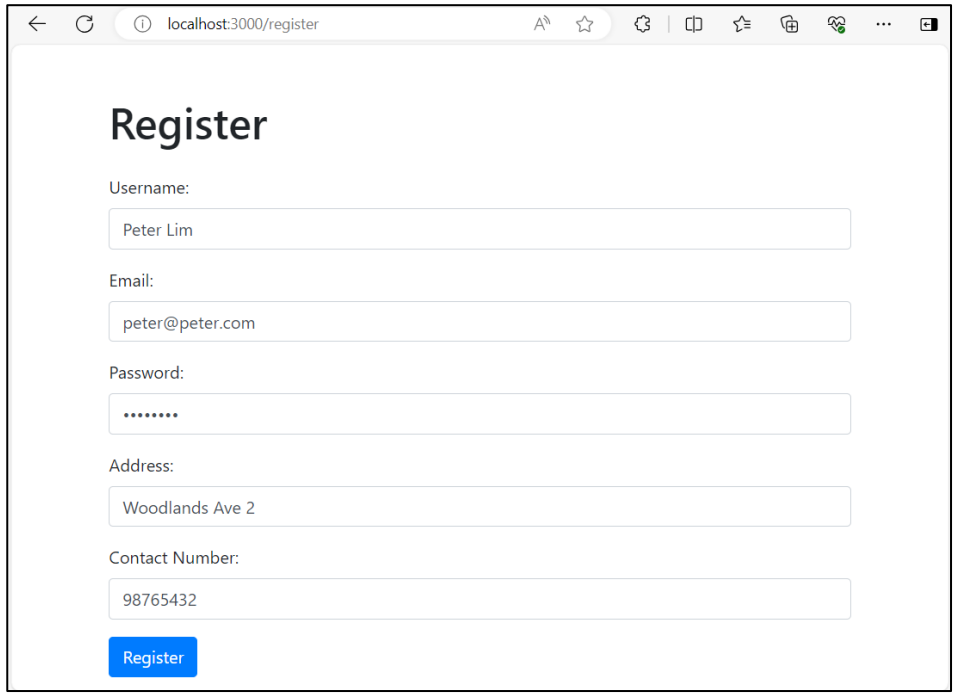
- Open app.js in Visual studio code and add in the code below:

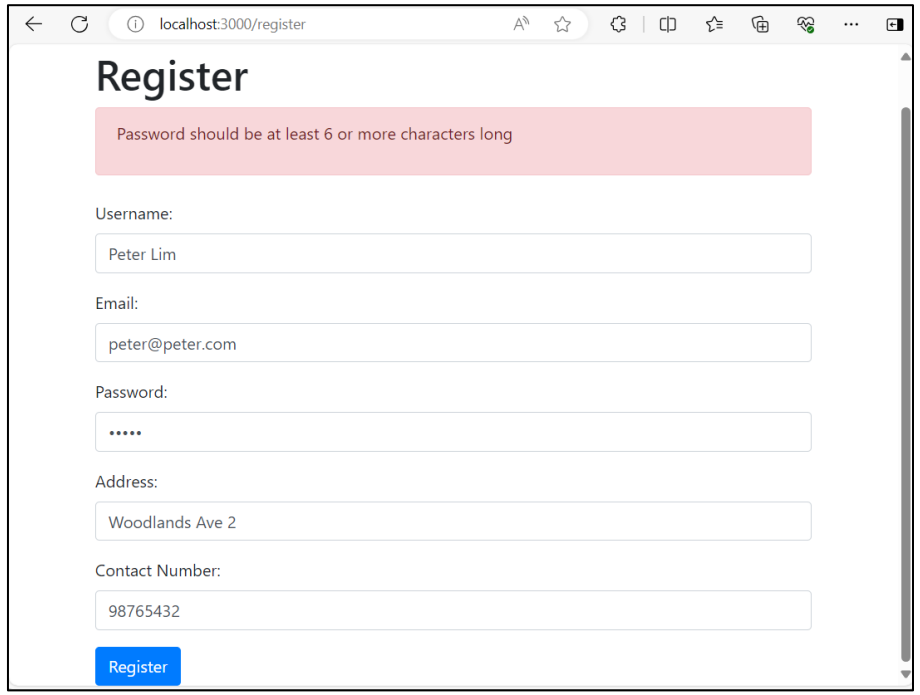
```
//******** TO DO: Integrate validateRegistration into the register route ********//
app.post('/register', validateRegistration, (req, res) => {

    const { username, email, password, address, contact } = req.body;

    const sql = 'INSERT INTO users (username, email, password, address, contact) VALUES (?, ?, SHA1(?), ?, ?)';
    db.query(sql, [username, email, password, address, contact], (err, result) => {
        if (err) {
            throw err;
        }
        console.log(result);
        req.flash('success', 'Registration successful! Please log in.');
        res.redirect('/login');
    });
});
```

This function checks for the presence of required fields and enforces a minimum password length (6). If any validation fails, it sends an appropriate response or redirects the user with an error message.

3. Test the validation by registering a new user.

- Run your app and try to create a new user.
- Try entering a password shorter than 6 characters and see what happens.





3. Test the validation by registering a new user.

- Run your app and try to create a new user.
- Enter all the information correctly and you should see a “Cannot Get /login” error. This is ok as we have not defined the /login route yet.

- Check your database and you should see the new user record inserted.

←T→			id	username	email	password	address	contact
☐	Edit	Copy	Delete	1	Peter Lim	peter@peter.com	9a4dc937ac9f581c535ea262096f0af7e8002fba	woodlands ave 2	98765432

Authentication

Authentication

- Authentication is the process of verifying the identity of a user.
- Involves verifying a username and password.
- User submits login form, server checks credentials, and establishes a session if valid.

Implement Basic Authentication

- Implement login function
- Steps:
  1. Set up express-session for session management. (Done earlier)
  2. Create login routes.
  3. Create logout route.
  4. Test the authentication process by logging in users.

2. Create login routes.

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

```
//******** TO DO: Insert code for login routes to render login page below ********//
app.get('/login', (req, res) => {

    res.render('login', {
        messages: req.flash('success'), // Retrieve success messages from the session and pass them to the view
        errors: req.flash('error') // Retrieve error messages from the session and pass them to the view
    });
});
```

2. Create login routes.

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

```
//******** TO DO: Insert code for login routes for form submission below ********//
app.post('/login', (req, res) => {
    const { email, password } = req.body;

    // Validate email and password
    if (!email || !password) {
        req.flash('error', 'All fields are required.');
        return res.redirect('/login');
    }

    const sql = 'SELECT * FROM users WHERE email = ? AND password = SHA1(?)';
    db.query(sql, [email, password], (err, results) => {
        if (err) {
```

*code continues next slide

2. Create login routes.

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

```
    const sql = 'SELECT * FROM users WHERE email = ? AND password = SHA1(?)';
    db.query(sql, [email, password], (err, results) => {
        if (err) {
            throw err;
        }

        if (results.length > 0) {
            // Successful login
            req.session.user = results[0]; // store user in session
            req.flash('success', 'Login successful!');
            res.redirect('/');
        } else {
            // Invalid credentials
            req.flash('error', 'Invalid email or password.');
            res.redirect('/login');
        }
    });
});
```

*code continues from here

Login Routes

```
    const sql = 'SELECT * FROM users WHERE email = ? AND password = SHA1(?)';
    db.query(sql, [email, password], (err, results) => {
        if (err) {
            throw err;
        }

        if (results.length > 0) {
            // Successful login
            req.session.user = results[0]; // store user in session
            req.flash('success', 'Login successful!');
            res.redirect('/');
        } else {
            // Invalid credentials
            req.flash('error', 'Invalid email or passwo
            res.redirect('/login');
        }
    });
});
```

app.js – /login route

app.js – / route

```
app.get('/', (req, res) => {
    res.render('index', { user: req.session.user , messages: req.flash('success')});
});
```

index.ejs

```
<% if (messages && messages.length > 0) { %>
    <div class="alert alert-success">
        <% messages.forEach(function(message) { %>
            <p><%= message %></p>
        <% }); %>
    </div>
    Hello , <%= user.username %>
    <br> <a href="/logout" class="btn btn-primary mt-3">Logout</a>
<% } else { %>
<a href="/register" class="btn btn-primary mt-3">Register</a>
<a href="/login" class="btn btn-primary mt-3">Login</a>
<% } %>
```

3. Create logout route.

- Open app.js (RegistrationApp) in Visual studio code and add in the code below:

```
//******** TO DO: Insert code for logout route ********//
app.get('/logout', (req, res) => {
    req.session.destroy();
    res.redirect('/');
});
```

- **req.session.destroy();:** The server destroys the user's session, removing all session data from the session store. This includes any information about the user's login status and other session-specific data.
- **res.redirect('/');:** The user is redirected to the home page (or another specified URL). This typically takes the user back to the main entry point of the application.

Let’s test it out

- Save all your files.
- Open your terminal and enter “npx nodemon app.js” (if your server is not already running.)
- Open your browser and enter the url http://localhost:3000 and test the authentication process by registering, logging in and logging out users.

Sample Screenshots (Register)

Mary Tan inserted into database

Username:

Mary Tan

Email:

mary@mary.com

Password:

•••••••

Address:

Tampines Ave 1

Contact Number:

87654321

Register

←T→	id	username	email	password	address	contact
Edit Copy Delete	1	Peter Lim	peter@peter.com	9a4dc937ac9f581c535ea262096f0af7e8002fba	woodlands ave 2	98765432
Edit Copy Delete	2	Mary Tan	mary@mary.com	ed253aa920d1c7d0b16c9bc85fe581af8286d88f	Tampines Ave 1	87654321

Login

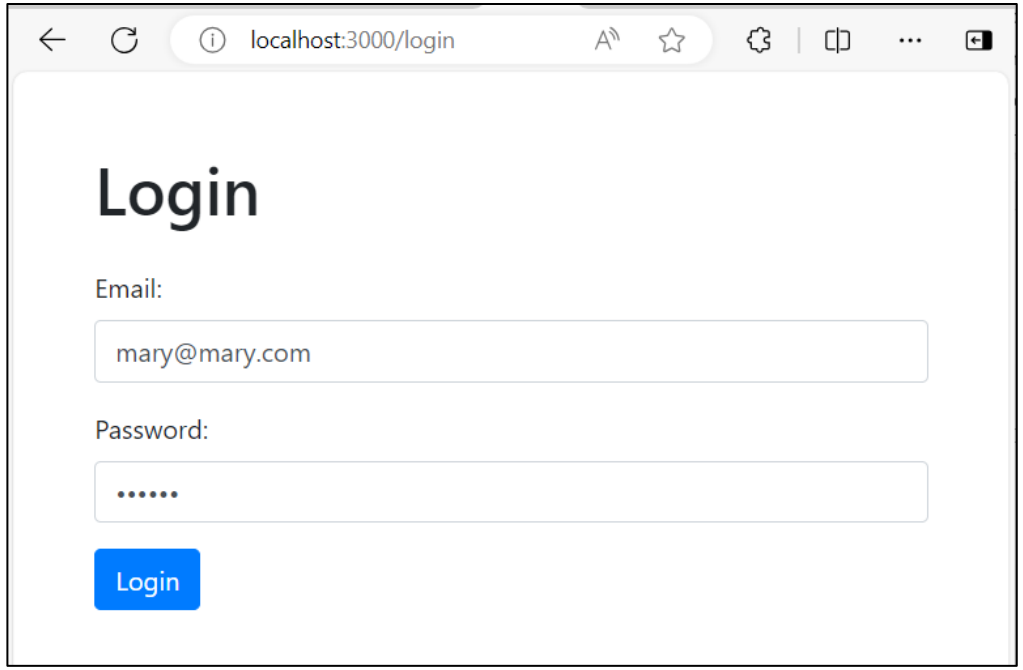
Registration successful! Please log in.

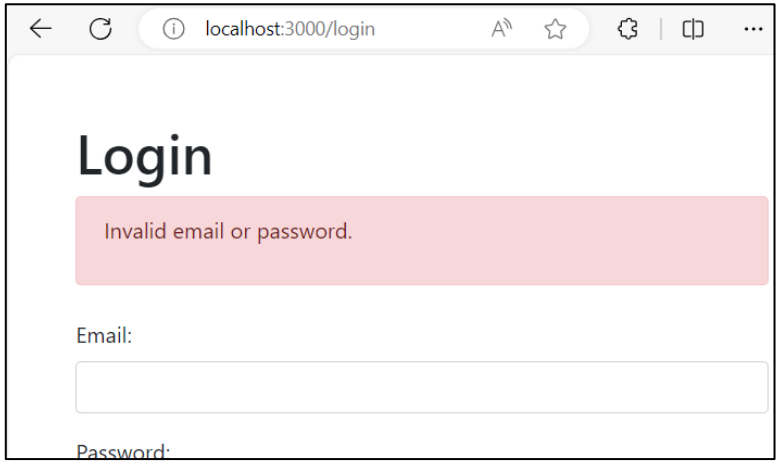
Email:

Password:

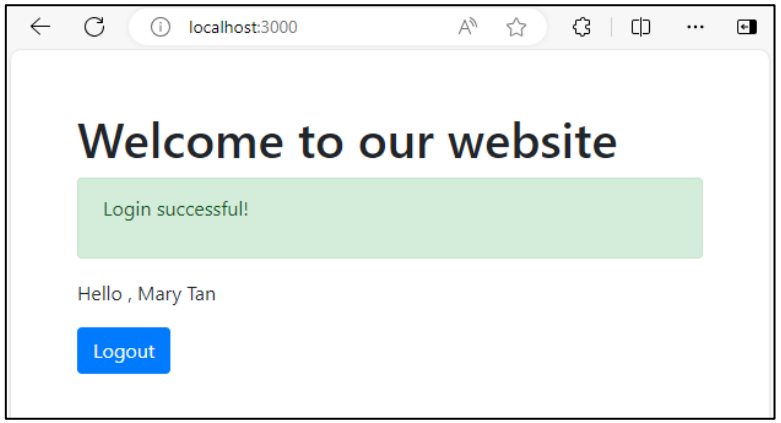
Login

Sample Screenshots (Login)



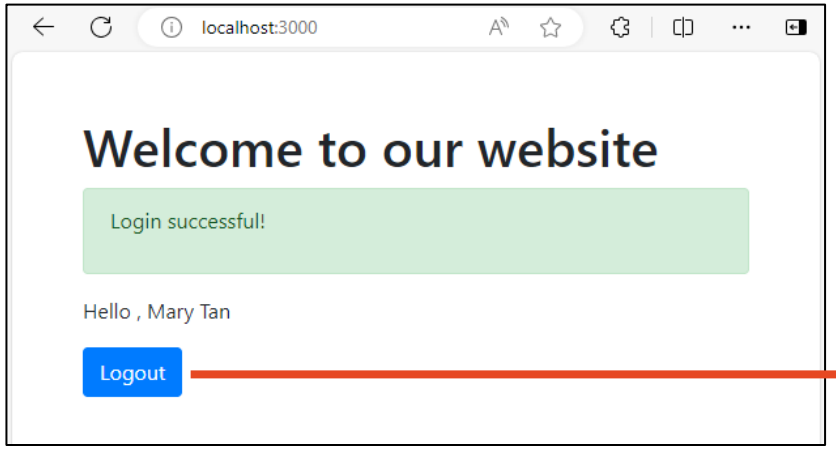


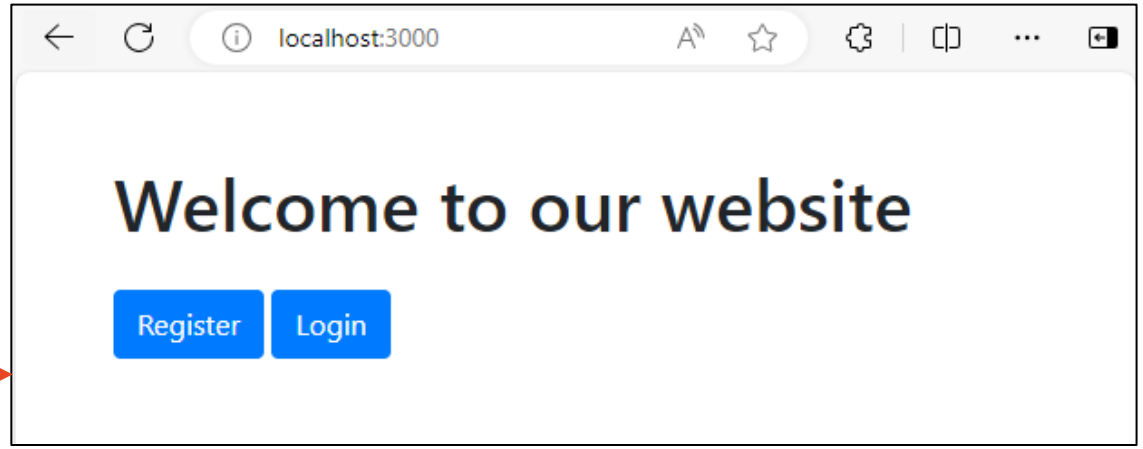
Login Failed



Login Success

Sample Screenshots (Logout)





What have you learnt?

- Form Validation
- Authentication