# University of Colombo School of Computing

## SCS 2208 - Rapid Application Development

*Lab Sheet 10*

---

**Express:**
Express is a web application framework for Node.js that simplifies the process of building robust and scalable web applications. It provides a set of tools for handling routing, handling requests and responses, and managing middleware.

**Express-session:**
Express-session is a middleware that enables the storage of session data on the server side. It allows you to manage user sessions and store session-related information, providing a way to maintain user state across multiple requests.

**Passport:**
Passport is an authentication middleware for Node.js. It simplifies the process of implementing user authentication strategies, such as OAuth, OpenID, and more. Passport provides a consistent and flexible way to authenticate users for your application.

**Passport-local:**
Passport-local is an extension of the Passport framework that focuses on local authentication, which means using username and password to authenticate users. It's commonly used for implementing basic authentication for web applications.

**Body-parser:**
The body-parser middleware is used to parse the incoming request body in various formats (such as JSON, URL-encoded, etc.) and make it accessible in the req.body object. This is essential for handling data submitted through forms or API requests.

**Multer:**
Multer is a middleware specifically designed for handling multipart/form-data, which is typically used for uploading files through forms. It simplifies the process of handling file uploads and allows you to access uploaded files in your request handlers.

**Path:**
The path module is a built-in Node.js module used for working with file paths and directories. It provides functions to manipulate file and directory paths in a platform-independent way, which is useful when dealing with file systems.

## Activities

01. You are tasked with creating a simple web application for user registration and authentication. The application should use Passport for authentication and Express Session for session management. Additionally, use Pug (or EJS) as the templating engine for rendering views.

    I. Set up the Express application with the necessary packages: `express`, `express-session`, `passport`, `passport-local`, `pug` (or `ejs`), and any other required packages.

    **Hints:**
    - Use passport-local and passport packages.
    - In the local strategy callback, call done(null, user) if authentication is successful, and done(null, false) if it fails.

    II. Create routes for the following functionalities:

- User registration: Users should be able to register with a username and password. Store the user's credentials securely using Passport.

- User login: Implement a login page where users can enter their credentials and log in. Use Passport to authenticate users.

- Home page: After successful login, users should see a welcome message displaying their username.

    **Hints:**

    - Mention that express-session helps maintain state across multiple HTTP requests.
    - Describe the process of creating and storing session data.
    - Emphasize that Passport uses the session to keep track of authenticated users.
    - Explain that the session data is stored on the server while a session ID is stored on the client.

02. Extend the previous web application to allow users to upload a profile picture during the registration process. Use Multer to handle file uploads and save the uploaded images in a designated directory.

I. Install and configure the `multer` package to handle file uploads. Set up Multer to store uploaded images in a directory called "uploads".

**Hints:**

- Use npm install multer to install Multer.
- Require multer at the top of your file.
- Define a storage variable using multer.diskStorage to configure where uploaded files should be stored.
- Use the destination and filename functions in the storage configuration.

II. Modify the user registration form to include a file input field for uploading profile pictures.

**Hints:**

- Inside your registration form's HTML, add a new input field of type "file" for profile picture uploads.
- Set the form's enctype attribute to "multipart/form-data" to handle file uploads.

III. Update the registration route to handle profile picture uploads. After successful registration, users should see their uploaded profile picture on the home page.

**Hints:**

- In your registration route, add the upload.single('profilePicture') middleware after the route path and before the route handler function.
- Inside the route handler, you can access the uploaded file using req.file.
- You'll need to handle the file storage and potentially save the file information in your user model or database.
- To display the uploaded profile picture on the home page, make sure to pass the file path to the view and use it in your template.