Step 1: Create User Authentication Routes

1.1 Create a Users Route File

1. **Create File:**

In your routes folder, create a file named users.js.

2. **Add Registration, Login, and Logout Endpoints:**

Paste the following code into routes/users.js:

```
import express from 'express';
import User from '../models/User.js';
import bcrypt from 'bcrypt';  // for password hashing

const router = express.Router();

// GET /users/register - Show registration form
router.get('/register', (req, res) => {
  res.render('users/register', { title: 'Register' });
});

// POST /users/register - Process registration form
router.post('/register', async (req, res) => {
  try {
    const { username, email, password } = req.body;
    // Hash the password before saving
    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({ username, email, password: hashedPassword });
    await user.save();
    res.redirect('/users/login');
  } catch (err) {
    console.error(err);
    res.status(500).send('Error during registration.');
  }
});

// GET /users/login - Show login form
router.get('/login', (req, res) => {
  res.render('users/login', { title: 'Login' });
});

// POST /users/login - Process login form
router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(401).send('Invalid credentials');
    }
    // Compare hashed password
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(401).send('Invalid credentials');
    }
    // Store user info in session (you might choose to store only minimal
info)
    req.session.user = { id: user._id, username: user.username, email:
user.email };
    res.redirect('/');
  } catch (err) {
    console.error(err);
    res.status(500).send('Error during login.');
  }
});

// GET /users/logout - Log out user
router.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      console.error(err);
      return res.status(500).send('Could not log out.');
    }
    res.redirect('/');
  });
});

export default router;
```

1.2 Install bcrypt

If you haven’t already installed bcrypt, run:

```
npm install bcrypt
```

1.3 Mount the Users Router

1. **Update app.js:**

Import and mount the new router. In your app.js file, add:

```
import usersRouter from './routes/users.js';
// ...
```

```
app.use('/users', usersRouter);
```

Step 2: Create EJS Views for User Authentication

2.1 Create the Users Views Folder

Inside views/, create a new folder called users.

2.2 Create the Registration Form

1. **Create File:**

Create views/users/register.ejs with the following content:

```
<%- include('../partials/header.ejs') %>
<div class="container my-4">
  <h2>Register</h2>
  <form action="/users/register" method="POST">
    <div class="mb-3">
      <label for="username" class="form-label">Username:</label>
      <input type="text" id="username" name="username" class="form-
control" required>
    </div>
    <div class="mb-3">
      <label for="email" class="form-label">Email:</label>
      <input type="email" id="email" name="email" class="form-control"
required>
    </div>
    <div class="mb-3">
      <label for="password" class="form-label">Password:</label>
      <input type="password" id="password" name="password" class="form-
control" required>
    </div>
    <button type="submit" class="btn btn-primary">Register</button>
  </form>
  <p class="mt-3">Already have an account? <a href="/users/login">Login
here</a>.</p>
</div>
<%- include('../partials/footer.ejs') %>
```

2.3 Create the Login Form

1. **Create File:**

Create views/users/login.ejs with the following content:

```
<%- include('../partials/header.ejs') %>
<div class="container my-4">
  <h2>Login</h2>
  <form action="/users/login" method="POST">
    <div class="mb-3">
      <label for="email" class="form-label">Email:</label>
      <input type="email" id="email" name="email" class="form-control"
required>
    </div>
    <div class="mb-3">
      <label for="password" class="form-label">Password:</label>
      <input type="password" id="password" name="password" class="form-
control" required>
    </div>
    <button type="submit" class="btn btn-primary">Login</button>
  </form>
  <p class="mt-3">Don't have an account? <a
href="/users/register">Register here</a>.</p>
</div>
<%- include('../partials/footer.ejs') %>
```

Step 3: Protect the Cart Route with Authentication Middleware

3.1 Create an Authentication Middleware

1. **Create Middleware File:**

In your project root (or a new folder called middleware), create a file named auth.js with the following content:

```
// middleware/auth.js
export function ensureAuthenticated(req, res, next) {
  if (req.session && req.session.user) {
    return next();
  }
  res.redirect('/users/login');
}
```

3.2 Apply the Middleware to the Cart Route

1. **Update routes/cart.js:**

At the top of routes/cart.js, import your middleware:

```
import { ensureAuthenticated } from '../middleware/auth.js';
```

2. **Protect the Cart GET Route:**

Update the cart route to require login:

```
// GET /cart - Display the shopping cart (protected)
router.get('/', ensureAuthenticated, (req, res) => {
  res.render('cart', { title: 'Your Cart', cart: req.session.cart });
});
```

You can optionally protect the POST endpoints as well if you want to restrict adding, updating, or removing items to logged-in users.

Step 4: Update the Navbar to Reflect Authentication Status

4.1 Modify the Header Partial

1. **Open views/partials/header.ejs** and update the navbar section to conditionally show login/register or logout links:

```
<!-- Inside the navbar's unordered list -->
<ul class="navbar-nav ms-auto">
  <li class="nav-item"><a class="nav-link" href="/">Home</a></li>
  <li class="nav-item"><a class="nav-link" href="/products">Products</a>
</li>
  <li class="nav-item"><a class="nav-link" href="/products/new">New
Product</a></li>
  <li class="nav-item"><a class="nav-link" href="/cart">Cart</a></li>
  <% if (typeof session !== 'undefined' && session.user) { %>
    <li class="nav-item"><a class="nav-link" href="/users/logout">Logout
(<%= session.user.username %>)</a></li>
  <% } else { %>
    <li class="nav-item"><a class="nav-link" href="/users/login">Login</a>
</li>
    <li class="nav-item"><a class="nav-link"
href="/users/register">Register</a></li>
  <% } %>
</ul>
```

2. **Expose the Session to All Views:**

To use session in your EJS templates, add a middleware in app.js before your routes:

```
app.use((req, res, next) => {
  res.locals.session = req.session;
  next();
});
```

Step 5: Final Testing & Verification

1. **Start Your Server:**

In your terminal, run:

```
node app.js
```

2. **Test the User Authentication Flow:**

• Navigate to http://localhost:3000/users/register to create a new user.

• Then go to http://localhost:3000/users/login to log in.

• After login, check that your navbar now displays a “Logout” link with your username.

3. **Test the Protected Cart Route:**

• Try accessing http://localhost:3000/cart when not logged in—it should redirect you to the login page.

• Once logged in, visit the cart page again to ensure you can see the cart contents.