Day 4

Here's your full 📅 **Day 4 – Authentication with JWT in React + .NET Core** schedule in your preferred format.

📅 Day 4 – JWT Authentication (Login + Register)

🎯 **Goal:** Implement secure login and registration using JWT in your .NET Core API and React frontend.

🕒 Time: ~5 hours

✅ Task 1: Add User Model and Auth Tables in Backend (~45 min)

🧠 Why it matters:

You need a `User` table to store credentials securely and manage authentication.

📘 Concepts explained:

- `User` model
- `DbSet<User>` in `ApplicationDbContext`
- Password hashing
- JWT = JSON Web Token

🔨 Example code:

```
// Models/User.cs
public class User {
  public int Id { get; set; }
  public string Email { get; set; }
  public string PasswordHash { get; set; }
}
```

```
// ApplicationDbContext.cs
public DbSet<User> Users { get; set; }
```

Then run:

```
dotnet ef migrations add AddUserTable
dotnet ef database update
```

✅ Task 2: Implement Register + Login Endpoints (~1 hr)

🧠 Why it matters:

Your API must securely issue a token when users log in — this token is used to protect private routes.

📘 Concepts explained:

- `Register` API: store user with hashed password
- `Login` API: check password and return JWT token
- `Microsoft.IdentityModel.Tokens`
- `SymmetricSecurityKey`, `JwtSecurityToken`

🔨 Example endpoint (simplified):

```
[HttpPost("register")]
public async Task<IActionResult> Register(UserDto dto) {
  var hashed = BCrypt.Net.BCrypt.HashPassword(dto.Password);
  var user = new User { Email = dto.Email, PasswordHash = hashed };
  _context.Users.Add(user);
  await _context.SaveChangesAsync();
  return Ok();
}

[HttpPost("login")]
public IActionResult Login(UserDto dto) {
  var user = _context.Users.SingleOrDefault(x => x.Email == dto.Email);
  if (user == null || !BCrypt.Net.BCrypt.Verify(dto.Password, user.Password
Hash))
    return Unauthorized();

  var token = CreateJwtToken(user);
  return Ok(new { token });
}
```

✅ Task 3: Configure JWT Middleware in Program.cs (~30 min)

🧠 Why it matters:

The middleware validates the token in each request's `Authorization` header.

📘 Concepts explained:

- `AddAuthentication().AddJwtBearer()`
- Secret key, token validation parameters

🔨 Sample:

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationSche
me)
  .AddJwtBearer(options => {
    options.TokenValidationParameters = new TokenValidationParameters {
      ValidateIssuer = false,
      ValidateAudience = false,
      ValidateLifetime = true,
      ValidateIssuerSigningKey = true,
      IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetByt
es("YourSuperSecretKey"))
    };
  });

app.UseAuthentication();
```

✅ Task 4: Create `Login.jsx` and `Register.jsx` in React (~1 hr)

🧠 Why it matters:

Your frontend needs login and registration pages to interact with the backend.

📘 Concepts explained:

- `useState` for form
- Axios POST requests
- Save token in `localStorage`

🔨 Example:

```
const handleLogin = async (e) => {
  e.preventDefault();
  const res = await axios.post('/auth/login', { email, password });
  localStorage.setItem('token', res.data.token);
};
```

✅ Task 5: Set up Protected Route + AuthContext (~1 hr)

🧠 Why it matters:

You'll restrict access to certain pages unless the user is logged in.

📘 Concepts explained:

- `React Context API` for storing auth state
- `ProtectedRoute` wrapper
- Read from `localStorage` on app load

🔨 Sample `ProtectedRoute.jsx`:

```
const ProtectedRoute = ({ children }) => {
  const token = localStorage.getItem('token');
  return token ? children : <Navigate to="/login" />;
};
```

✅ Task 6: Push Changes to GitHub (~15 min)

🧠 Why it matters:

All progress should be tracked and ready to be showcased.

🔨 Git:

```
git add .
git commit -m "feat: added JWT auth in backend and login/register in front
end"
git push
```

✅ By End of Day 4, You Will Have:

Feature	Status
User model + DB setup	✅
Register/Login endpoints	✅
JWT issuance + validation	✅
React login/register pages	✅
Auth token saved in client	✅
Protected routes	✅
Code pushed to GitHub	✅

Let me know once you're done so we can start **Day 5: Filtering, sorting, and search UI for tasks**! 🔍📊