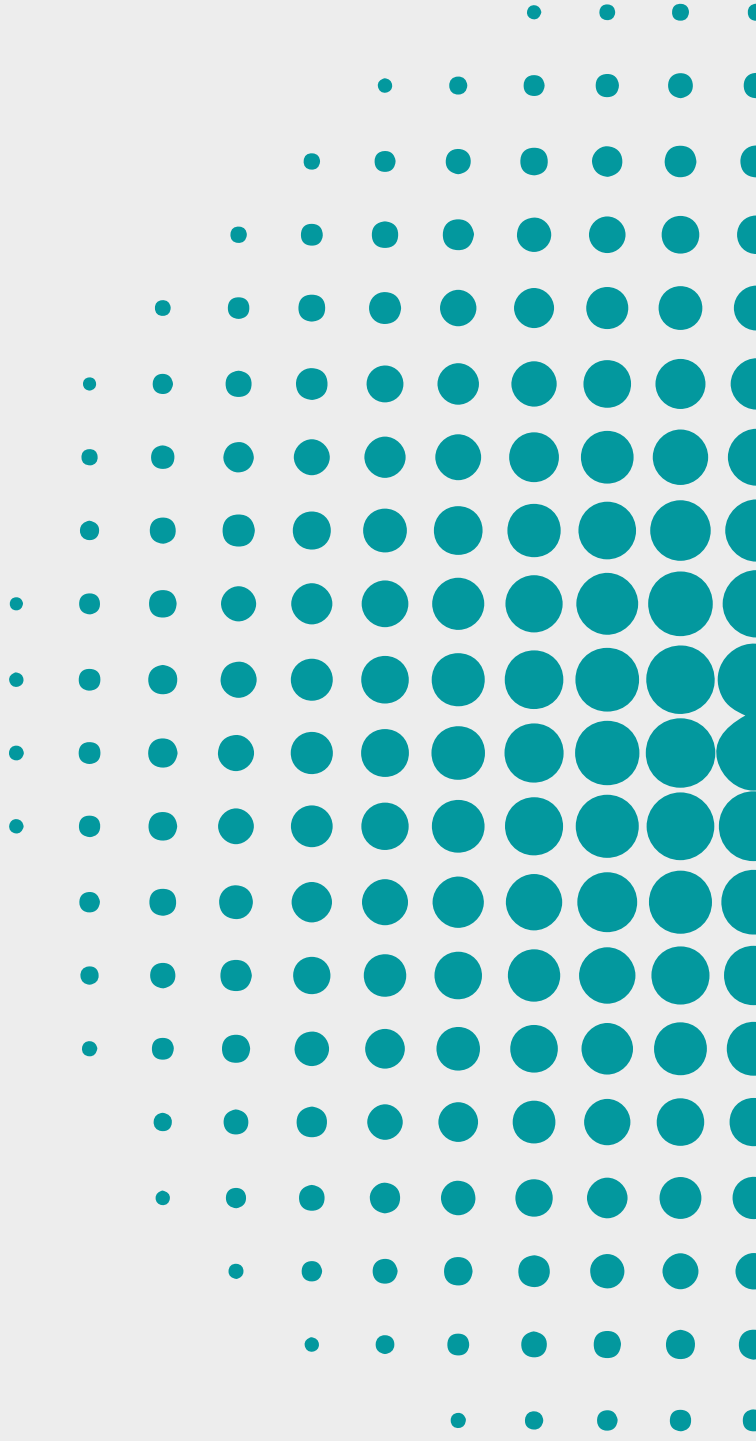
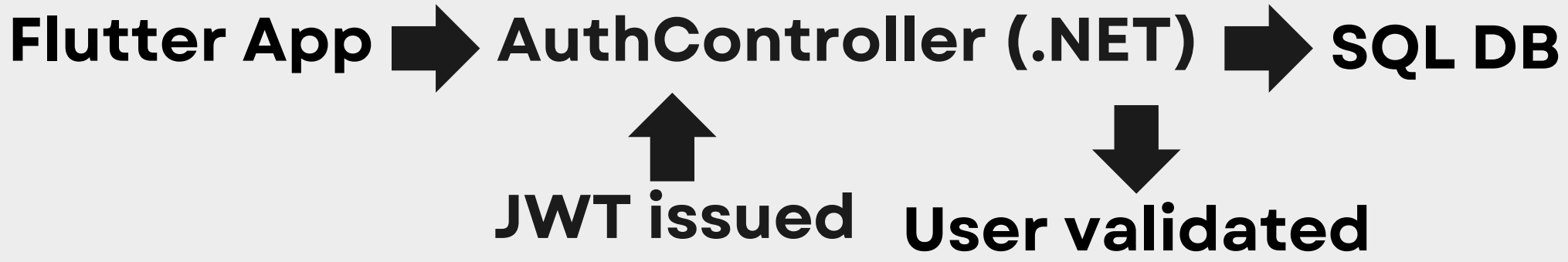
JWT Flow



App Architecture Overview



Login Method

```
[HttpPost("login")]
0 references
public IActionResult Login([FromBody] LoginRequest request)
{
    // Check for missing username or password
    if (string.IsNullOrWhiteSpace(request.Username) || string.IsNullOrWhiteSpace(request.Password))
        return BadRequest("Username and password are required");

    // Try to find the user in the database
    var user = _context.User.FirstOrDefault(u => u.Username == request.Username);
    if (user == null)
        return Unauthorized("Invalid credentials");

    // Verify that the password is correct
    var result = _passwordHasher.VerifyHashedPassword(user, user.PasswordHash, request.Password);
    if (result == PasswordVerificationResult.Failed)
        return Unauthorized("Invalid credentials");
```

JWT Creation

```
var claims = new List<Claim>
{
    new Claim(ClaimTypes.Name, user.Username),
    new Claim(ClaimTypes.Role, roleName),
    new Claim("UserId", user.UserId.ToString())
};

// Create a JWT key and sign it
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_conf
var creds = new SigningCredentials(key, SecurityAlgorithms.Hmac

// Create the JWT token
var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Issuer"],
    audience: _configuration["Jwt:Audience"],
    claims: claims,
    expires: DateTime.Now.AddHours(1),
    signingCredentials: creds);

// Return token + user info to the frontend
return Ok(new
{
    token = new JwtSecurityTokenHandler().WriteToken(token),
    user = new { user.UserId, user.Username, Role = roleName }
});
```

Register Method

```
[HttpPost("register")]
0 references
public IActionResult Register([FromBody] RegisterRequest model)
{
    // Check for empty fields
    if (string.IsNullOrWhiteSpace(model.Username) || string.IsNullOrWhiteSpace(model.Password) || string.IsNullOrWhiteSpace(model.Role))
        return BadRequest("All fields are required");

    // Check if username already exists
    if (_context.User.Any(u => u.Username == model.Username))
        return BadRequest("Username already exists");

    // Find the matching role in the Role table
    var role = _context.Role.FirstOrDefault(r => r.Name == model.Role);
    if (role == null)
        return BadRequest("Invalid role");

    // Create new user with hashed password and role
    var user = new UserModel
    {
        Username = model.Username,
        PasswordHash = _passwordHasher.HashPassword(null, model.Password),
        RoleId = role.RoleId
    };

    // Save the new user to the database
    _context.User.Add(user);
    _context.SaveChanges();

    // Create claims just like in login
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.Name, user.Username),
        new Claim(ClaimTypes.Role, role.Name),
        new Claim("UserId", user.UserId.ToString())
    };

    // Build the JWT token
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]!));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
```

Secure Password Handling with Hashing

Program.cs

```
builder.Services.AddScoped<IPasswordHasher<UserModel>, PasswordHasher<UserModel>>();
```

- AddScoped<TInterface, TImplementation>()

→ This tells ASP.NET Core:

“Every time a new web request comes in, give a fresh instance of PasswordHasher<UserModel> whenever IPasswordHasher<UserModel> is requested.”

- IPasswordHasher<UserModel>

→ This is the interface (like a contract). Controller depends on this type.

- PasswordHasher<UserModel>

→ This is the actual implementation of the password hashing logic using a secure algorithm (PBKDF2 by default in .NET).

AuthContoller.cs

```
PasswordHash = _passwordHasher.HashPassword(null, model.Password),
```

- It takes the plain-text password from the user.
- Then it uses the injected IPasswordHasher<UserModel> to hash it securely.
- The resulting PasswordHash is what’s saved to the database – not the plain password.

Register Flow

```
Future<bool> login(String username, String password) async {
  final response = await http.post(
    Uri.parse('$baseUrl/Login'),
    headers: {'Content-Type': 'application/json'},
    body: jsonEncode({
      "username": username,
      "password": password,
    }),
```

- Sends user input (username, password, role) to https://localhost:7106/auth/register
- This hits ourbackend's [HttpPost("register")] endpoint
- Backend hashes our password
- Creates JWT token & returns it with user info

Register Flow

```dart
if (response.statusCode == 200) {
  final body = jsonDecode(response.body);
  final prefs = await SharedPreferences.getInstance();

  final token = body['token'];
  print("JWT Token: $token");

  await prefs.setString('jwt', token);
  await prefs.setString('role', body['user']['role']);
```

- Check if login was successful (statusCode == 200)
- Decode the JSON response from backend
- Extract the JWT token and user data
- Save token and role using SharedPreferences
- Enables secure local storage for session persistence
- Navigates to HomePage using saved info

JWT Protected Route Test

```dart
Future<void> testProtectedEndpoint() async {
  final prefs = await SharedPreferences.getInstance();
  final token = prefs.getString('jwt');

  final response = await http.get(
    Uri.parse('http://10.0.2.2:5000/user/hello'),
    headers: {
      'Authorization': 'Bearer $token',
      'Content-Type': 'application/json',
    },
  );

  if (response.statusCode == 200) {
    _showSnackBar("Backend says: ${response.body}");
  } else {
    _showSnackBar("Failed: ${response.statusCode}");
  }
}
```

- **Calls** the /user/hello route in the backend.
- **Sends** JWT token in the Authorization header.
- **If the** token is valid, backend returns a greeting.
- **Confirms** secure routes are protected by authentication

Authentication Flow Summary

- User enters username & password in Flutter app
- AuthService sends credentials to .NET AuthController
- Backend hashes password, checks database, and returns JWT
- JWT is stored locally using SharedPreferences
- JWT is used to access protected routes

Thank You

Sofiia Yanytska