```
 1  using DatabaseConnect;
 2  using Microsoft.AspNetCore.Authorization;
 3  using Microsoft.AspNetCore.Mvc;
 4  using Microsoft.EntityFrameworkCore;
 5  using Microsoft.Extensions.Configuration;
 6  using Microsoft.Extensions.Logging;
 7  using Microsoft.IdentityModel.Tokens;
 8  using System;
 9  using System.Collections.Generic;
10  using System.Linq;
11  using System.Security.Claims;
12  using System.Threading.Tasks;
13  using static LibraryAppMVC.Models.Models;
14  using DatabaseConnect.Entities;
15  using System.IdentityModel.Tokens.Jwt;
16  using System.Text;
17  using Microsoft.AspNetCore.Cryptography.KeyDerivation;
18
19  namespace LibraryAppMVC.Controllers
20  {
21      [Route("/user/")] // All endpoints checked 2/25/18, logout not working but
          not important (token dumped client side at logout)
22      public class UserController : Controller
23      {
24          private IConfiguration _config;
25          private Context _ctx;
26          private readonly ILogger _logger;
27
28          public UserController(IConfiguration config, Context context,
              ILogger<UserController> logger)
29          {
30              _config = config;
31              _ctx = context;
32              _logger = logger;
33          }
34
35
36          [Route("login")]
37          [AllowAnonymous]
38          [HttpPost]
39          public IActionResult CreateToken([FromBody]LoginModel login) // Checked
              2/24/18 working
40          {
41              IActionResult response = Unauthorized();
42              var user = Authenticate(login);
43
44              if (user != null)
45              {
46                  response = BuildToken(user);
47              }
48              return response;
49          }
```

```
50
51          [Route("logout")]
52          [Authorize]
53          [HttpPost]
54          public IActionResult Logout() // Checked 2/24/18 NOT working TODO, maybe
              not important because client dumps token on logout
55          {
56              string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
57              int userID = _ctx.Users
58                  .Single(u => u.SchoolID == schoolID)
59                  .UserID;
60              _ctx.Users
61                  .Single(u => u.UserID == userID);
62              _ctx.SaveChanges();
63              return Ok();
64          }
65
66          [Route("info")]
67          [Authorize]
68          [HttpGet]
69          public IActionResult UserInfo()
70          {
71              string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
72              var user = _ctx.Users
73                  .Single(u => u.SchoolID == schoolID);
74              int userID = user.UserID;
75
76              var checkouts = _ctx.Checkouts
77                  .Where(c => c.Active)
78                  .Where(c => c.UserID == userID)
79                  .Include(c => c.Book)
80                  .ToList();
81
82              var reservations = _ctx.Reservations
83                  .Where(r => r.Active)
84                  .Where(r => r.UserID == userID)
85                  .Include(r => r.Book)
86                  .ToList();
87
88              foreach (Checkout c in checkouts)
89              {
90                  c.User = null;
91              }
92              foreach (Reservation r in reservations)
93              {
94                  r.User = null;
95              }
96              user.PasswordHash = null;
97              user.Salt = null;
98              var resp = new { checkouts, reservations, user };
```

```
 99              return Json(resp);
100          }
101
102          private IActionResult BuildToken(UserModel user)
103          {
104              var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config
                   ["Jwt:Key"]));
105              var creds = new SigningCredentials(key,
                   SecurityAlgorithms.HmacSha256);
106              var claims = new[]
107              {
108                  new Claim(JwtRegisteredClaimNames.Sub, user.StudentID),
109                  new Claim(JwtRegisteredClaimNames.Jti, user.TokenVersion.ToString
                       ())
110              };
111
112              var token = new JwtSecurityToken(
113                  _config["Jwt:Issuer"],
114                  _config["Jwt:Issuer"],
115                  expires: DateTime.Now.AddMinutes(Convert.ToDouble(_config
                       ["LoginDurationMinutes"])),
116                  signingCredentials: creds,
117                  claims: claims
118                  );
119              return Ok(
120                  new
121                  {
122                      token = new JwtSecurityTokenHandler().WriteToken(token),
123                      expiration = token.ValidTo
124                  });
125          }
126
127          private UserModel Authenticate(LoginModel login)
128          {
129              User User;
130              UserModel usermodel = null;
131              try
132              {
133                  User = _ctx.Users
134                      .Single(u => u.SchoolID.Equals(login.Username));
135              }
136              catch
137              {
138                  return null;  // No user found with specified school ID
139              }
140              if (VerifyPass(login.Password, User.Salt, User.PasswordHash))
141              {
142                  usermodel = new UserModel { Name = User.FullName, StudentID =
                       User.SchoolID, TokenVersion = User.TokenVersion };
143              }
144              return usermodel;
145          }
```

```
146
147         private Boolean VerifyPass(String RawPass, String Salt, String
              PasswordHash)
148         {
149             byte[] salt_array = Convert.FromBase64String(Salt);
150             String hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
151                 password: RawPass,
152                 salt: salt_array,
153                 prf: KeyDerivationPrf.HMACSHA1,
154                 iterationCount: 10000,
155                 numBytesRequested: 256 / 8));
156             return hashed.Equals(PasswordHash);
157         }
158     }
159 }
160
```