

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

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

```
99         .ToList();
100
101         foreach(Checkout c in checkouts)
102         {
103             c.User = null;
104         }
105         foreach(Reservation r in reservations)
106         {
107             r.User = null;
108         }
109         user.PasswordHash = null;
110         user.Salt = null;
111         var resp = new { checkouts, reservations, user };
112         return Json(resp);
113     }
114
115     private IActionResult BuildToken(UserModel user)
116     {
117         var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config
118             ["Jwt:Key"]));
119         var creds = new SigningCredentials(key,
120             SecurityAlgorithms.HmacSha256);
121         var claims = new[]
122         {
123             new Claim(JwtRegisteredClaimNames.Sub, user.StudentID),
124             new Claim(JwtRegisteredClaimNames.Jti, user.TokenVersion.ToString
125                 ())
126         };
127
128         var token = new JwtSecurityToken(
129             _config["Jwt:Issuer"],
130             _config["Jwt:Issuer"],
131             expires: DateTime.Now.AddMinutes(Convert.ToDouble(_config
132                 ["LoginDurationMinutes"])),
133             signingCredentials: creds,
134             claims: claims
135         );
136         return Ok(
137             new {
138                 token = new JwtSecurityTokenHandler().WriteToken(token),
139                 expiration = token.ValidTo
140             });
141     }
142
143     private UserModel Authenticate(LoginModel login)
144     {
145         User user;
146         UserModel usermodel = null;
147         try
148         {
149             user = _ctx.Users
150                 .Single(u => u.SchoolID.Equals(login.Username));
```

```
147     }
148     catch
149     {
150         return null; // No user found with specified school ID
151     }
152     if(VerifyPass(login.Password, User.Salt, User.PasswordHash))
153     {
154         usermodel = new UserModel { Name = User.FullName, StudentID =      ↗
155             User.SchoolID, TokenVersion = User.TokenVersion };
156     }
157     return usermodel;
158 }
159 private Boolean VerifyPass(String RawPass, String Salt, String      ↗
160     PasswordHash)
161 {
162     byte[] salt_array = Convert.FromBase64String(Salt);
163     String hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
164         password: RawPass,
165         salt: salt_array,
166         prf: KeyDerivationPrf.HMACSHA1,
167         iterationCount: 10000,
168         numBytesRequested: 256 / 8));
169     return hashed.Equals(PasswordHash);
170 }
171
172
173 [Route("/library/")] // All endpoints checked 2/25/18
174 public class LibraryController : Controller
175 {
176     private Context _ctx;
177
178     public LibraryController(Context context)
179     {
180         _ctx = context;
181     }
182
183
184     [Route("checkout")]
185     [HttpPost]
186     [Authorize]
187     public IActionResult BookCheckout([FromBody]TransactionRequest      ↗
188         request) // Checked 2/24/18 working
189     {
190         string schoolID = User.Claims.FirstOrDefault(c => c.Type ==      ↗
191             ClaimTypes.NameIdentifier).Value;
192         int userID = _ctx.Users
193             .Single(u => u.SchoolID == schoolID)
194             .UserID;
195
196         if(!_ctx.Books.Any(b => b.BookID == request.BookID))
```

```

195     {
196         return StatusCode(409, "Book does not exist");
197     }
198
199     int limit = _ctx.UserUType_rel // Get max checked out books for
        usertype
200         .Include(ut => ut.UType)
201         .Single(ut => ut.UserID == userID)
202         .UType
203         .CheckoutLimit;
204
205     int current = _ctx.Checkouts // Get current user checked out books
206         .Where(c => c.Active)
207         .Where(c => c.UserID == userID)
208         .Count();
209
210     if (current >= limit) // Check to see if user can checkout more
        books
211     {
212         return StatusCode(409, $"You already have checked out {current}
        books, as many as you can.");
213     }
214
215     bool CheckedOut = _ctx.Checkouts
216         .Where(c => c.Active && c.BookID.Equals(request.BookID))
217         .Count() > 0;
218
219     if (CheckedOut)
220     {
221         return StatusCode(409, "Already checked out");
222     }
223
224     _ctx.Checkouts
225         .Add(new Checkout { BookID = request.BookID, UserID = userID,
        Active=true, CheckoutDate=DateTime.Now,
        DueDate=DateTime.Now.AddDays(14) });
226     _ctx.SaveChanges();
227     return Ok();
228 }
229
230 [Route("checkin")]
231 [HttpPost]
232 [Authorize]
233 public IActionResult BookCheckin([FromBody]TransactionRequest request) //
        Checked 2/24/18 working
234 {
235     string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
        ClaimTypes.NameIdentifier).Value;
236     int userID = _ctx.Users
237         .Single(u => u.SchoolID == schoolID)
238         .UserID;
239

```

```
240         if (!_ctx.Books.Any(b => b.BookID == request.BookID))
241         {
242             return StatusCode(409, "Book does not exist");
243         }
244
245         _ctx.Checkouts
246             .Where(c => c.BookID == request.BookID && c.UserID == userID)
247             .Last()
248             .Active = false;
249         _ctx.SaveChanges();
250         return Ok();
251     }
252
253     [Route("reserve")]
254     [HttpPost]
255     [Authorize]
256     public IActionResult ReserveBook([FromBody]TransactionRequest request) // ↗
257     {
258         string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
259             ClaimTypes.NameIdentifier).Value;
260         int userID = _ctx.Users
261             .Single(u => u.SchoolID == schoolID)
262             .UserID;
263
264         if (!_ctx.Books.Any(b => b.BookID == request.BookID))
265         {
266             return StatusCode(409, "Book does not exist");
267         }
268
269         Boolean BookAvailable = _ctx.Checkouts
270             .Where(c => c.BookID == request.BookID && c.Active)
271             .Count() > 0;
272
273         BookAvailable = true;
274
275         Boolean UserAlreadyReserved = _ctx.Reservations
276             .Where(r => r.Active && r.UserID == userID)
277             .Count() > 0;
278
279         if(!UserAlreadyReserved || !BookAvailable)
280         {
281             _ctx.Reservations
282                 .Add(new Reservation { BookID = request.BookID, UserID =
283                     userID, Datetime = DateTime.Now, Active = true});
284             _ctx.SaveChanges();
285             return Ok();
286         }
287         else if(UserAlreadyReserved)
288         {
289             return StatusCode(409, "You have already reserved this book");
290         }
291     }
```

```
289         else if(BookAvailable)
290         {
291             return StatusCode(409, "This book can be checked out now, not reserved");
292         }
293         return StatusCode(500);
294     }
295 }
296
297 [Route("fill_reservation")]
298 [HttpPost]
299 [Authorize]
300 public IActionResult FillReservation([FromBody]TransactionRequest request) // Checked 2/25/18 working
301 {
302     string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
303     int userID = _ctx.Users
304         .Single(u => u.SchoolID == schoolID)
305         .UserID;
306
307     if (!_ctx.Books.Any(b => b.BookID == request.BookID))
308     {
309         return StatusCode(409, "Book does not exist");
310     }
311
312     Boolean CheckedOut = _ctx.Checkouts
313         .Single(c => c.BookID == request.BookID)
314         .Active == true;
315
316     if(!CheckedOut)
317     {
318         IActionResult resp = BookCheckout(request);
319         _ctx.Reservations
320             .Where(r => r.Active && r.BookID.Equals(request.BookID) && r.UserID.Equals(userID))
321             .OrderByDescending(r => r.Datetime)
322             .First()
323             .Active = false;
324         _ctx.SaveChanges();
325         return resp;
326     }
327     else
328     {
329         return StatusCode(409, "Book already checked out");
330     }
331 }
332
333 [Route("renew")]
334 [HttpPost]
335 [Authorize]
336 public IActionResult RenewBook([FromBody]TransactionRequest request) //
```

Checked 2/25/18 working

```

337     {
338         string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
339             ClaimTypes.NameIdentifier).Value;
340         int userID = _ctx.Users
341             .Single(u => u.SchoolID == schoolID)
342             .UserID;
343         bool AlreadyReserved = _ctx.Reservations
344             .Where(r => r.Active && r.BookID.Equals(request.BookID))
345             .Count() > 0;
346         bool OverRenewals = _ctx.Checkouts
347             .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
348                 c.UserID.Equals(userID))
349             .OrderByDescending(c => c.CheckoutDate)
350             .First()
351             .Renewals > 2;
352         if(AlreadyReserved || OverRenewals)
353         {
354             return Forbid();
355         }
356         DateTime Checkout = _ctx.Checkouts
357             .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
358                 c.UserID.Equals(userID))
359             .OrderByDescending(c => c.CheckoutDate)
360             .First()
361             .CheckoutDate;
362         _ctx.Checkouts
363             .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
364                 c.UserID.Equals(userID))
365             .OrderByDescending(c => c.CheckoutDate)
366             .First()
367             .Renewals += 1;
368         _ctx.SaveChanges();
369         return Ok();
370     }
371 }
372
373
374
375
376 [Route("/simple/")] // All endpoints checked 2/25/18
377 public class SimpleController : Controller
378 {
379     private Context _ctx;
380
381     public SimpleController(Context context)
382     {

```



```
383     _ctx = context;
384 }
385
386 [AllowAnonymous]
387 [Route("books")]
388 [HttpGet]
389 public IActionResult GetABook(string title, int page = 1) // Checked 7
    2/25/18 working
390 {
391     List<Book> a;
392     if (title != null) // Title specified
393     {
394         a = _ctx.Books
395             .Where(b => b.Title.Contains(title))
396             .Include(book => book.Cover)
397             .Include(book => book.AuthorBooks)
398             .ThenInclude(ab => ab.Author)
399             .ToList();
400     }
401     else // Title not specified
402     {
403         if (page < 1) { page = 1; }
404         int pos_i = (page - 1) * 10;
405         int pos_f = page * 10;
406         int count = _ctx.Books.Count();
407         if (pos_f > count) { pos_f = count; }
408         if (pos_i > count) { a = new List<Book>(); }
409         else
410         {
411             a = _ctx.Books
412                 .Include(book => book.Cover)
413                 .Include(book => book.AuthorBooks)
414                 .ThenInclude(ab => ab.Author)
415                 .ToList()
416                 .GetRange(pos_i, (pos_f - pos_i));
417         }
418     }
419
420     foreach(Book b in a)
421     {
422         List<String> AuthorList = new List<String>();
423         //b.Cover.Books = null;
424         foreach(AuthorBook ab in b.AuthorBooks)
425         {
426             AuthorList.Add(ab.Author.Name);
427         }
428         b.Authors = AuthorList;
429         b.AuthorBooks = null;
430     }
431     return Json(a);
432 }
433
```

```
434 [Route("checkouts")]
435 [AllowAnonymous]
436 [HttpGet]
437 public IActionResult GetCheckouts() // Checked 2/25/18 working
438 {
439     var CheckoutList = _ctx.Checkouts
440         .Include(c => c.Book)
441         .Where(c => c.Active)
442         .ToList();
443     return Json(CheckoutList);
444 }
445
446 [Route("reservations")]
447 [AllowAnonymous]
448 [HttpGet]
449 public IActionResult GetReservations() // Checked 2/25/18 working
450 {
451     var CheckoutList = _ctx.Reservations
452         .Include(r => r.Book)
453         .Where(r => r.Active)
454         .ToList();
455     return Json(CheckoutList);
456 }
457 }
458
459
460 [Route("/dev/")] // All endpoints checked 2/25/18
461 public class DevController : Controller
462 {
463     private Context _ctx;
464     public DevController(Context context)
465     {
466         _ctx = context;
467     }
468
469     [Route("adduser")]
470     [HttpPost]
471     public IActionResult AddUser([FromBody]NewUser newuser) // Checked 2/25/18 working
472     {
473         User user = new User() { SchoolID = newuser.Username, Password =
474             newuser.Password };
475         byte[] salt = new byte[128 / 8];
476         using (var rng = RandomNumberGenerator.Create())
477         {
478             rng.GetBytes(salt);
479         }
480         string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
481             password: user.Password,
482             salt: salt,
483             prf: KeyDerivationPrf.HMACSHA1,
484             iterationCount: 10000,
```

```
484         numBytesRequested: 256 / 8));
485         user.Salt = Convert.ToBase64String(salt);
486         user.PasswordHash = hashed;
487         _ctx.Users.Add(user);
488         _ctx.SaveChanges();
489         int UserID = _ctx.Users
490             .Single(u => u.SchoolID == user.SchoolID)
491             .UserID;
492         _ctx.UserUType_rel
493             .Add(new UserUType { UserID = UserID, UTypeID = 1 });
494         _ctx.SaveChanges();
495         return Ok();
496     }
497 }
498 }
499
```