```
 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.Diagnostics;
13  using System.IdentityModel.Tokens.Jwt;
14  using System.IO;
15  using System.Linq;
16  using System.Net;
17  using System.Net.Http;
18  using System.Security.Claims;
19  using System.Security.Cryptography;
20  using System.Text;
21  using System.Threading.Tasks;
22  using System.Web;
23
24  namespace LibraryAppMVC.Controllers
25  {
26      [Route("")]
27      [ApiExplorerSettings(IgnoreApi = true)]
28      public class SwaggerRedirectController : Controller
29      {
30          [Route("")]
31          [HttpGet]
32          [ApiExplorerSettings(IgnoreApi = true)]
33          public IActionResult RedirectToSwaggerUi()
34          {
35              return Redirect("swagger");
36          }
37      }
38
39      [Route("/user/")] // All endpoints checked 2/25/18, logout not working but
          not important (token dumped client side at logout)
40      public class UserController : Controller
41      {
42          private IConfiguration _config;
43          private Context _ctx;
44          private readonly ILogger _logger;
45
46          public UserController(IConfiguration config, Context context,
              ILogger<UserController> logger)
47          {
48              _config = config;
49              _ctx = context;
50              _logger = logger;
```

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

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

```
147             UserModel usermodel = null;
148             try
149             {
150                 User = _ctx.Users
151                     .Single(u => u.SchoolID.Equals(login.Username));
152             }
153             catch
154             {
155                 return null;  // No user found with specified school ID
156             }
157             if(VerifyPass(login.Password, User.Salt, User.PasswordHash))
158             {
159                 usermodel = new UserModel { Name = User.FullName, StudentID =
                      User.SchoolID, TokenVersion = User.TokenVersion };
160             }
161             return usermodel;
162         }
163
164         private Boolean VerifyPass(String RawPass, String Salt, String
              PasswordHash)
165         {
166             byte[] salt_array = Convert.FromBase64String(Salt);
167             String hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
168                 password: RawPass,
169                 salt: salt_array,
170                 prf: KeyDerivationPrf.HMACSHA1,
171                 iterationCount: 10000,
172                 numBytesRequested: 256 / 8));
173             return hashed.Equals(PasswordHash);
174         }
175
176         public class LoginModel
177         {
178             public String Username { get; set; }
179             public String Password { get; set; }
180         }
181
182         public class UserModel
183         {
184             public String Name { get; set; }
185             public String StudentID { get; set; }
186             public int TokenVersion { get; set; }
187         }
188
189
190     }
191
192
193     [Route("/library/")] // All endpoints checked 2/25/18
194     public class LibraryController : Controller
195     {
196         private Context _ctx;
```

```
197
198         public LibraryController(Context context)
199         {
200             _ctx = context;
201         }
202
203         public class TransactionRequest
204         {
205             public int BookID { get; set; }
206         }
207
208         [Route("checkout")]
209         [HttpPost]
210         [Authorize]
211         public IActionResult BookCheckout([FromBody]TransactionRequest
              request) // Checked 2/24/18 working
212         {
213             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
214             int userID = _ctx.Users
215                 .Single(u => u.SchoolID == schoolID)
216                 .UserID;
217
218             if(!_ctx.Books.Any(b => b.BookID == request.BookID))
219             {
220                 return StatusCode(409, "Book does not exist");
221             }
222
223             int limit = _ctx.UserUType_rel // Get max checked out books for
                  usertype
224                 .Include(ut => ut.UType)
225                 .Single(ut => ut.UserID == userID)
226                 .UType
227                 .CheckoutLimit;
228
229             int current = _ctx.Checkouts // Get current user checked out books
230                 .Where(c => c.Active)
231                 .Where(c => c.UserID == userID)
232                 .Count();
233
234             if (current >= limit)  // Check to see if user can checkout more
                  books
235             {
236                 return StatusCode(409, $"You already have checked out {current}
                      books, as many as you can.");
237             }
238
239             bool CheckedOut = _ctx.Checkouts
240                 .Where(c => c.Active && c.BookID.Equals(request.BookID))
241                 .Count() > 0;
242
243             if (CheckedOut)
```

```
244             {
245                 return StatusCode(409, "Already checked out");
246             }
247
248             _ctx.Checkouts
249                 .Add(new Checkout { BookID = request.BookID, UserID = userID,
                      Active=true, CheckoutDate=DateTime.Now,
                      DueDate=DateTime.Now.AddDays(14) });
250             _ctx.SaveChanges();
251             return Ok();
252         }
253
254         [Route("checkin")]
255         [HttpPost]
256         [Authorize]
257         public IActionResult BookCheckin([FromBody]TransactionRequest request) // Checked 2/24/18 working
258         {
259             string schoolID = User.Claims.FirstOrDefault(c => c.Type == 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             _ctx.Checkouts
270                 .Where(c => c.BookID == request.BookID && c.UserID == userID)
271                 .Last()
272                 .Active = false;
273             _ctx.SaveChanges();
274             return Ok();
275         }
276
277         [Route("reserve")]
278         [HttpPost]
279         [Authorize]
280         public IActionResult ReserveBook([FromBody]TransactionRequest request) // Checked 2/25/18 working
281         {
282             string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
283             int userID = _ctx.Users
284                 .Single(u => u.SchoolID == schoolID)
285                 .UserID;
286
287             if (!_ctx.Books.Any(b => b.BookID == request.BookID))
288             {
289                 return StatusCode(409, "Book does not exist");
```

```
290             }
291
292             Boolean BookAvailable = _ctx.Checkouts
293                 .Where(c => c.BookID == request.BookID && c.Active)
294                 .Count() > 0;
295
296             BookAvailable = true;
297
298             Boolean UserAlreadyReserved = _ctx.Reservations
299                 .Where(r => r.Active && r.UserID == userID)
300                 .Count() > 0;
301
302             if(!UserAlreadyReserved || !BookAvailable)
303             {
304                 _ctx.Reservations
305                     .Add(new Reservation { BookID = request.BookID, UserID =
                        userID, Datetime = DateTime.Now, Active = true});
306                 _ctx.SaveChanges();
307                 return Ok();
308             }
309             else if(UserAlreadyReserved)
310             {
311                 return StatusCode(409, "You have already reserved this book");
312             }
313             else if(BookAvailable)
314             {
315                 return StatusCode(409, "This book can be checked out now, not
                      reserved");
316             }
317             return StatusCode(500);
318
319         }
320
321         [Route("fill_reservation")]
322         [HttpPost]
323         [Authorize]
324         public IActionResult FillReservation([FromBody]TransactionRequest
              request) // Checked 2/25/18 working
325         {
326             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
327             int userID = _ctx.Users
328                 .Single(u => u.SchoolID == schoolID)
329                 .UserID;
330
331             if (!_ctx.Books.Any(b => b.BookID == request.BookID))
332             {
333                 return StatusCode(409, "Book does not exist");
334             }
335
336             Boolean CheckedOut = _ctx.Checkouts
337                 .Single(c => c.BookID == request.BookID)
```

```
338                 .Active == true;
339
340             if(!CheckedOut)
341             {
342                 IActionResult resp = BookCheckout(request);
343                 _ctx.Reservations
344                     .Where(r => r.Active && r.BookID.Equals(request.BookID) &&
                        r.UserID.Equals(userID))
345                     .OrderByDescending(r => r.Datetime)
346                     .First()
347                     .Active = false;
348                 _ctx.SaveChanges();
349                 return resp;
350             }
351             else
352             {
353                 return StatusCode(409, "Book already checked out");
354             }
355         }
356
357         [Route("renew")]
358         [HttpPost]
359         [Authorize]
360         public IActionResult RenewBook([FromBody]TransactionRequest request) //
            Checked 2/25/18 working
361         {
362             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                ClaimTypes.NameIdentifier).Value;
363             int userID = _ctx.Users
364                 .Single(u => u.SchoolID == schoolID)
365                 .UserID;
366
367             bool AlreadyReserved = _ctx.Reservations
368                 .Where(r => r.Active && r.BookID.Equals(request.BookID))
369                 .Count() > 0;
370             bool OverRenewals = _ctx.Checkouts
371                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
372                 .OrderByDescending(c => c.CheckoutDate)
373                 .First()
374                 .Renewals > 2;
375             if(AlreadyReserved || OverRenewals)
376             {
377                 return Forbid();
378             }
379             DateTime Checkout = _ctx.Checkouts
380                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
381                 .OrderByDescending(c => c.CheckoutDate)
382                 .First()
383                 .CheckoutDate;
384             _ctx.Checkouts
```

```
385                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
386                 .OrderByDescending(c => c.CheckoutDate)
387                 .First()
388                 .CheckoutDate = Checkout.AddDays(7);
389             _ctx.Checkouts
390                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
391                 .OrderByDescending(c => c.CheckoutDate)
392                 .First()
393                 .Renewals += 1;
394             _ctx.SaveChanges();
395             return Ok();
396         }
397     }
398
399
400     [Route("/simple/")] // All endpoints checked 2/25/18
401     public class SimpleController : Controller
402     {
403         private Context _ctx;
404
405         public SimpleController(Context context)
406         {
407             _ctx = context;
408         }
409
410         [AllowAnonymous]
411         [Route("books")]
412         [HttpGet]
413         public IActionResult GetABook(string title, int page = 1) // Checked
              2/25/18 working
414         {
415             List<Book> a;
416             if (title != null)  // Title specified
417             {
418                 a = _ctx.Books
419                     .Where(b => b.Title.Contains(title))
420                     .Include(book => book.Cover)
421                     .Include(book => book.AuthorBooks)
422                         .ThenInclude(ab => ab.Author)
423                     .ToList();
424             }
425             else  // Title not specified
426             {
427                 if (page < 1) { page = 1; }
428                 int pos_i = (page - 1) * 10;
429                 int pos_f = page * 10;
430                 int count = _ctx.Books.Count();
431                 if (pos_f > count) { pos_f = count; }
432                 if (pos_i > count) { a = new List<Book>(); }
433                 else
```

```
434                 {
435                     a = _ctx.Books
436                         .Include(book => book.Cover)
437                         .Include(book => book.AuthorBooks)
438                             .ThenInclude(ab => ab.Author)
439                         .ToList()
440                         .GetRange(pos_i, (pos_f - pos_i));
441                 }
442             }
443 
444             foreach(Book b in a)
445             {
446                 List<String> AuthorList = new List<String>();
447                 //b.Cover.Books = null;
448                 foreach(AuthorBook ab in b.AuthorBooks)
449                 {
450                     AuthorList.Add(ab.Author.Name);
451                 }
452                 b.Authors = AuthorList;
453                 b.AuthorBooks = null;
454             }
455             return Json(a);
456         }
457 
458         [Route("checkouts")]
459         [AllowAnonymous]
460         [HttpGet]
461         public IActionResult GetCheckouts() // Checked 2/25/18 working
462         {
463             var CheckoutList = _ctx.Checkouts
464                 .Include(c => c.Book)
465                 .Where(c => c.Active)
466                 .ToList();
467             return Json(CheckoutList);
468         }
469 
470         [Route("reservations")]
471         [AllowAnonymous]
472         [HttpGet]
473         public IActionResult GetReservations() // Checked 2/25/18 working
474         {
475             var CheckoutList = _ctx.Reservations
476                 .Include(r => r.Book)
477                 .Where(r => r.Active)
478                 .ToList();
479             return Json(CheckoutList);
480         }
481     }
482 
483 
484     [Route("/dev/")] // All endpoints checked 2/25/18
485     public class DevController : Controller
```

```
486     {
487         private Context _ctx;
488         public DevController(Context context)
489         {
490             _ctx = context;
491         }
492 
493         public class NewUser
494         {
495             public string Username { get; set; }
496             public string Password { get; set; }
497         }
498 
499         [Route("adduser")]
500         [HttpPost]
501         public IActionResult AddUser([FromBody]NewUser newuser) // Checked 2/25/18 working
502         {
503             User user = new User() { SchoolID = newuser.Username, Password = newuser.Password };
504             byte[] salt = new byte[128 / 8];
505             using (var rng = RandomNumberGenerator.Create())
506             {
507                 rng.GetBytes(salt);
508             }
509             string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
510                     password: user.Password,
511                     salt: salt,
512                     prf: KeyDerivationPrf.HMACSHA1,
513                     iterationCount: 10000,
514                     numBytesRequested: 256 / 8));
515             user.Salt = Convert.ToBase64String(salt);
516             user.PasswordHash = hashed;
517             _ctx.Users.Add(user);
518             _ctx.SaveChanges();
519             int UserID = _ctx.Users
520                 .Single(u => u.SchoolID == user.SchoolID)
521                 .UserID;
522             _ctx.UserUType_rel
523                 .Add(new UserUType { UserID = UserID, UTypeID = 1 });
524             _ctx.SaveChanges();
525             return Ok();
526         }
527     }
528 }
529 
```