```
 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
          not important (token dumped client side at logout)
35      public class UserController : Controller
36      {
37          private IConfiguration _config;
38          private Context _ctx;
39          private readonly ILogger _logger;
40
41          public UserController(IConfiguration config, Context context,
              ILogger<UserController> logger)
42          {
43              _config = config;
44              _ctx = context;
45              _logger = logger;
46          }
47
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
                   ["Jwt:Key"]));
118              var creds = new SigningCredentials(key,
                   SecurityAlgorithms.HmacSha256);
119              var claims = new[]
120              {
121                  new Claim(JwtRegisteredClaimNames.Sub, user.StudentID),
122                  new Claim(JwtRegisteredClaimNames.Jti, user.TokenVersion.ToString
                       ())
123              };
124
125              var token = new JwtSecurityToken(
126                  _config["Jwt:Issuer"],
127                  _config["Jwt:Issuer"],
128                  expires: DateTime.Now.AddMinutes(Convert.ToDouble(_config
                       ["LoginDurationMinutes"])),
129                  signingCredentials: creds,
130                  claims: claims
131                  );
132              return Ok(
133                  new {
134                      token = new JwtSecurityTokenHandler().WriteToken(token),
135                      expiration = token.ValidTo
136                  });
137          }
138
139          private UserModel Authenticate(LoginModel login)
140          {
141              User User;
142              UserModel usermodel = null;
143              try
144              {
145                  User = _ctx.Users
146                      .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 =
                      User.SchoolID, TokenVersion = User.TokenVersion };
155             }
156             return usermodel;
157         }
158
159         private Boolean VerifyPass(String RawPass, String Salt, String
              PasswordHash)
160         {
161             byte[] salt_array = Convert.FromBase64String(Salt);
162             String hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
163                 password: RawPass,
164                 salt: salt_array,
165                 prf: KeyDerivationPrf.HMACSHA1,
166                 iterationCount: 10000,
167                 numBytesRequested: 256 / 8));
168             return hashed.Equals(PasswordHash);
169         }
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
              request) // Checked 2/24/18 working
188         {
189             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
190             int userID = _ctx.Users
191                 .Single(u => u.SchoolID == schoolID)
192                 .UserID;
193
194             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) // ↵
               Checked 2/25/18 working
257         {
258             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==          ↵
                  ClaimTypes.NameIdentifier).Value;
259             int userID = _ctx.Users
260                 .Single(u => u.SchoolID == schoolID)
261                 .UserID;
262 
263             if (!_ctx.Books.Any(b => b.BookID == request.BookID))
264             {
265                 return StatusCode(409, "Book does not exist");
266             }
267 
268             Boolean BookAvailable = _ctx.Checkouts
269                 .Where(c => c.BookID == request.BookID && c.Active)
270                 .Count() > 0;
271 
272             Boolean UserAlreadyReserved = _ctx.Reservations
273                 .Where(r => r.Active && r.UserID == userID)
274                 .Count() > 0;
275 
276             if(!UserAlreadyReserved || !BookAvailable)
277             {
278                 _ctx.Reservations
279                     .Add(new Reservation { BookID = request.BookID, UserID =     ↵
                          userID, Datetime = DateTime.Now, Active = true});
280                 _ctx.SaveChanges();
281                 return Ok();
282             }
283             else if(UserAlreadyReserved)
284             {
285                 return StatusCode(409, "You have already reserved this book");
286             }
287             else if(BookAvailable)
288             {
```

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

```
335             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
336             int userID = _ctx.Users
337                 .Single(u => u.SchoolID == schoolID)
338                 .UserID;
339
340             bool AlreadyReserved = _ctx.Reservations
341                 .Where(r => r.Active && r.BookID.Equals(request.BookID))
342                 .Count() > 0;
343             bool OverRenewals = _ctx.Checkouts
344                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
345                 .OrderByDescending(c => c.CheckoutDate)
346                 .First()
347                 .Renewals > 2;
348             if(AlreadyReserved || OverRenewals)
349             {
350                 return Forbid();
351             }
352             DateTime Checkout = _ctx.Checkouts
353                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
354                 .OrderByDescending(c => c.CheckoutDate)
355                 .First()
356                 .CheckoutDate;
357             _ctx.Checkouts
358                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
359                 .OrderByDescending(c => c.CheckoutDate)
360                 .First()
361                 .CheckoutDate = Checkout.AddDays(7);
362             _ctx.Checkouts
363                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
364                 .OrderByDescending(c => c.CheckoutDate)
365                 .First()
366                 .Renewals += 1;
367             _ctx.SaveChanges();
368             return Ok();
369         }
370     }
371
372
373     [Route("/simple/")] // All endpoints checked 2/25/18
374     public class SimpleController : Controller
375     {
376         private Context _ctx;
377
378         public SimpleController(Context context)
379         {
380             _ctx = context;
381         }
```

```
382
383         [Route("books")]
384         [AllowAnonymous]
385         [HttpGet]
386         public IActionResult GetABook(string title, int page = 1) // Checked
              2/25/18 working
387         {
388             List<Book> a;
389             if (title != null)  // Title specified
390             {
391                 a = _ctx.Books
392                     .Where(b => b.Title.Contains(title))
393                     .Include(book => book.AuthorBooks)
394                         .ThenInclude(ab => ab.Author)
395                     .ToList();
396             }
397             else  // Title not specified
398             {
399                 if (page < 1) { page = 1; }
400                 int pos_i = (page - 1) * 10;
401                 int pos_f = page * 10;
402                 int count = _ctx.Books.Count();
403                 if (pos_f > count) { pos_f = count; }
404                 if (pos_i > count) { a = new List<Book>(); }
405                 else
406                 {
407                     a = _ctx.Books
408                         .Include(book => book.AuthorBooks)
409                             .ThenInclude(ab => ab.Author)
410                         .ToList()
411                         .GetRange(pos_i, (pos_f - pos_i));
412                 }
413             }
414
415             foreach(Book b in a)
416             {
417                 List<String> AuthorList = new List<String>();
418                 foreach(AuthorBook ab in b.AuthorBooks)
419                 {
420                     AuthorList.Add(ab.Author.Name);
421                 }
422                 b.Authors = AuthorList;
423                 b.AuthorBooks = null;
424             }
425             return Json(a);
426         }
427
428         [Route("checkouts")]
429         [AllowAnonymous]
430         [HttpGet]
431         public IActionResult GetCheckouts() // Checked 2/25/18 working
432         {
```

```
433             var CheckoutList = _ctx.Checkouts
434                 .Include(c => c.Book)
435                 .Where(c => c.Active)
436                 .ToList();
437             return Json(CheckoutList);
438         }
439 
440         [Route("reservations")]
441         [AllowAnonymous]
442         [HttpGet]
443         public IActionResult GetReservations() // Checked 2/25/18 working
444         {
445             var CheckoutList = _ctx.Reservations
446                 .Include(r => r.Book)
447                 .Where(r => r.Active)
448                 .ToList();
449             return Json(CheckoutList);
450         }
451     }
452 
453 
454     [Route("/dev/")] // All endpoints checked 2/25/18
455     public class DevController : Controller
456     {
457         private Context _ctx;
458         public DevController(Context context)
459         {
460             _ctx = context;
461         }
462 
463         [Route("adduser")]
464         [HttpPost]
465         public IActionResult AddUser([FromBody]NewUser newuser) // Checked
            2/25/18 working
466         {
467             if (newuser.UserTypeInt == 0) { newuser.UserTypeInt = 1; }
468             User user = new User() { SchoolID = newuser.Username, Password =
                newuser.Password };
469             byte[] salt = new byte[128 / 8];
470             using (var rng = RandomNumberGenerator.Create())
471             {
472                 rng.GetBytes(salt);
473             }
474             string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
475                     password: user.Password,
476                     salt: salt,
477                     prf: KeyDerivationPrf.HMACSHA1,
478                     iterationCount: 10000,
479                     numBytesRequested: 256 / 8));
480             user.Salt = Convert.ToBase64String(salt);
481             user.PasswordHash = hashed;
482             _ctx.Users.Add(user);
```

```
483             _ctx.SaveChanges();
484             int UserID = _ctx.Users
485                 .Single(u => u.SchoolID == user.SchoolID)
486                 .UserID;
487             _ctx.UserUType_rel
488                 .Add(new UserUType { UserID = UserID, UTypeID = 1 });
489             _ctx.SaveChanges();
490             return Ok();
491         }
492     }
493 }
494
```