

```
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         Boolean UserAlreadyReserved = _ctx.Reservations
274             .Where(r => r.Active && r.UserID == userID)
275             .Count() > 0;
276
277         if(!UserAlreadyReserved || !BookAvailable)
278         {
279             _ctx.Reservations
280                 .Add(new Reservation { BookID = request.BookID, UserID =
281                     userID, Datetime = DateTime.Now, Active = true});
282             _ctx.SaveChanges();
283             return Ok();
284         }
285         else if(UserAlreadyReserved)
286         {
287             return StatusCode(409, "You have already reserved this book");
288         }
289         else if(BookAvailable)
290         {
291         }
```

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

```
335     string schoolID = User.Claims.FirstOrDefault(c => c.Type ==  
336         ClaimTypes.NameIdentifier).Value;  
337     int userID = _ctx.Users  
338         .Single(u => u.SchoolID == schoolID)  
339         .UserID;  
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) &&  
345             c.UserID.Equals(userID))  
346         .OrderByDescending(c => c.CheckoutDate)  
347         .First()  
348         .Renewals > 2;  
349     if(AlreadyReserved || OverRenewals)  
350     {  
351         return Forbid();  
352     }  
353     DateTime Checkout = _ctx.Checkouts  
354         .Where(c => c.Active && c.BookID.Equals(request.BookID) &&  
355             c.UserID.Equals(userID))  
356         .OrderByDescending(c => c.CheckoutDate)  
357         .First()  
358         .CheckoutDate;  
359     _ctx.Checkouts  
360         .Where(c => c.Active && c.BookID.Equals(request.BookID) &&  
361             c.UserID.Equals(userID))  
362         .OrderByDescending(c => c.CheckoutDate)  
363         .First()  
364         .CheckoutDate = Checkout.AddDays(7);  
365     _ctx.Checkouts  
366         .Where(c => c.Active && c.BookID.Equals(request.BookID) &&  
367             c.UserID.Equals(userID))  
368         .OrderByDescending(c => c.CheckoutDate)  
369         .First()  
370         .Renewals += 1;  
371     _ctx.SaveChanges();  
372     return Ok();  
373 }  
374  
375 [Route("/simple/")] // All endpoints checked 2/25/18  
376 public class SimpleController : Controller  
377 {  
378     private Context _ctx;  
379     public SimpleController(Context context)  
380     {  
381         _ctx = context;  
382     }  
383 }
```



```
382
383     [AllowAnonymous]
384     [Route("books")]
385     [HttpGet]
386     public IActionResult GetABook(string title, int page = 1) // Checked 7
387         2/25/18 working
388     {
389         List<Book> a;
390         if (title != null) // Title specified
391         {
392             a = _ctx.Books
393                 .Where(b => b.Title.Contains(title))
394                 .Include(book => book.Cover)
395                 .Include(book => book.AuthorBooks)
396                 .ThenInclude(ab => ab.Author)
397                 .ToList();
398         }
399         else // Title not specified
400         {
401             if (page < 1) { page = 1; }
402             int pos_i = (page - 1) * 10;
403             int pos_f = page * 10;
404             int count = _ctx.Books.Count();
405             if (pos_f > count) { pos_f = count; }
406             if (pos_i > count) { a = new List<Book>(); }
407             else
408             {
409                 a = _ctx.Books
410                     .Include(book => book.Cover)
411                     .Include(book => book.AuthorBooks)
412                     .ThenInclude(ab => ab.Author)
413                     .ToList()
414                     .GetRange(pos_i, (pos_f - pos_i));
415             }
416         }
417         foreach(Book b in a)
418         {
419             List<String> AuthorList = new List<String>();
420             //b.Cover.Books = null;
421             foreach(AuthorBook ab in b.AuthorBooks)
422             {
423                 AuthorList.Add(ab.Author.Name);
424             }
425             b.Authors = AuthorList;
426             b.AuthorBooks = null;
427         }
428         return Json(a);
429     }
430
431     [Route("checkouts")]
432     [AllowAnonymous]
```

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

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