```
 1  using DatabaseConnect;
 2  using DatabaseConnect.Entities;
 3  using Microsoft.AspNetCore.Authorization;
 4  using Microsoft.AspNetCore.Mvc;
 5  using Microsoft.EntityFrameworkCore;
 6  using System;
 7  using System.Collections.Generic;
 8  using System.Linq;
 9  using System.Security.Claims;
10  using System.Threading.Tasks;
11  using static LibraryAppMVC.Models.Models;
12
13  namespace LibraryAppMVC.Controllers
14  {
15      [Route("/library/")] // All endpoints checked 2/25/18
16      public class LibraryController : Controller
17      {
18          private Context _ctx;
19
20          public LibraryController(Context context)
21          {
22              _ctx = context;
23          }
24
25
26          [Route("checkout")]
27          [HttpPost]
28          [Authorize]
29          public IActionResult BookCheckout([FromBody]TransactionRequest
              request) // Checked 2/24/18 working
30          {
31              string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
32              int userID = _ctx.Users
33                  .Single(u => u.SchoolID == schoolID)
34                  .UserID;
35
36              if (!_ctx.Books.Any(b => b.BookID == request.BookID))
37              {
38                  return StatusCode(409, "Book does not exist");
39              }
40
41              int limit = _ctx.UserUType_rel // Get max checked out books for
                    usertype
42                  .Include(ut => ut.UType)
43                  .Single(ut => ut.UserID == userID)
44                  .UType
45                  .CheckoutLimit;
46
47              int current = _ctx.Checkouts // Get current user checked out books
48                  .Where(c => c.Active)
49                  .Where(c => c.UserID == userID)
```

```
50                  .Count();
51
52              if (current >= limit)  // Check to see if user can checkout more
                  books
53              {
54                  return StatusCode(409, $"You already have checked out {current}
                      books, as many as you can.");
55              }
56
57              bool CheckedOut = _ctx.Checkouts
58                  .Where(c => c.Active && c.BookID.Equals(request.BookID))
59                  .Count() > 0;
60
61              if (CheckedOut)
62              {
63                  return StatusCode(409, "Already checked out");
64              }
65
66              _ctx.Checkouts
67                  .Add(new Checkout { BookID = request.BookID, UserID = userID,
                      Active = true, CheckoutDate = DateTime.Now, DueDate =
                      DateTime.Now.AddDays(14) });
68              _ctx.SaveChanges();
69              return Ok();
70          }
71
72          [Route("checkin")]
73          [HttpPost]
74          [Authorize]
75          public IActionResult BookCheckin([FromBody]TransactionRequest request) //
                Checked 2/24/18 working
76          {
77              string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                    ClaimTypes.NameIdentifier).Value;
78              int userID = _ctx.Users
79                  .Single(u => u.SchoolID == schoolID)
80                  .UserID;
81
82              if (!_ctx.Books.Any(b => b.BookID == request.BookID))
83              {
84                  return StatusCode(409, "Book does not exist");
85              }
86
87              _ctx.Checkouts
88                  .Where(c => c.BookID == request.BookID && c.UserID == userID)
89                  .Last()
90                  .Active = false;
91              _ctx.SaveChanges();
92              return Ok();
93          }
94
95          [Route("reserve")]
```

```
96          [HttpPost]
97          [Authorize]
98          public IActionResult ReserveBook([FromBody]TransactionRequest request) // Checked 2/25/18 working
99          {
100             string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
101             int userID = _ctx.Users
102                 .Single(u => u.SchoolID == schoolID)
103                 .UserID;
104
105             if (!_ctx.Books.Any(b => b.BookID == request.BookID))
106             {
107                 return StatusCode(409, "Book does not exist");
108             }
109
110             Boolean BookAvailable = _ctx.Checkouts
111                 .Where(c => c.BookID == request.BookID && c.Active)
112                 .Count() > 0;
113
114             Boolean UserAlreadyReserved = _ctx.Reservations
115                 .Where(r => r.Active && r.UserID == userID)
116                 .Count() > 0;
117
118             if (!UserAlreadyReserved || !BookAvailable)
119             {
120                 _ctx.Reservations
121                     .Add(new Reservation { BookID = request.BookID, UserID = userID, Datetime = DateTime.Now, Active = true });
122                 _ctx.SaveChanges();
123                 return Ok();
124             }
125             else if (UserAlreadyReserved)
126             {
127                 return StatusCode(409, "You have already reserved this book");
128             }
129             else if (BookAvailable)
130             {
131                 return StatusCode(409, "This book can be checked out now, not reserved");
132             }
133             return StatusCode(500);
134
135         }
136
137         [Route("fill_reservation")]
138         [HttpPost]
139         [Authorize]
140         public IActionResult FillReservation([FromBody]TransactionRequest request) // Checked 2/25/18 working
141         {
142             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
```

```
                ClaimTypes.NameIdentifier).Value;
143             int userID = _ctx.Users
144                 .Single(u => u.SchoolID == schoolID)
145                 .UserID;
146
147             if (!_ctx.Books.Any(b => b.BookID == request.BookID))
148             {
149                 return StatusCode(409, "Book does not exist");
150             }
151
152             Boolean CheckedOut = _ctx.Checkouts
153                 .Any(c => c.BookID == request.BookID && c.UserID == userID &&
                      c.Active == true);
154
155             if (!CheckedOut)
156             {
157                 IActionResult resp = BookCheckout(request);
158                 _ctx.Reservations
159                     .Where(r => r.Active && r.BookID.Equals(request.BookID) &&
                          r.UserID.Equals(userID))
160                     .OrderByDescending(r => r.Datetime)
161                     .First()
162                     .Active = false;
163                 _ctx.SaveChanges();
164                 return resp;
165             }
166             else
167             {
168                 return StatusCode(409, "Book already checked out");
169             }
170         }
171
172         [Route("renew")]
173         [HttpPost]
174         [Authorize]
175         public IActionResult RenewBook([FromBody]TransactionRequest request) //
              Checked 2/25/18 working
176         {
177             string schoolID = User.Claims.FirstOrDefault(c => c.Type ==
                  ClaimTypes.NameIdentifier).Value;
178             int userID = _ctx.Users
179                 .Single(u => u.SchoolID == schoolID)
180                 .UserID;
181
182             bool AlreadyReserved = _ctx.Reservations
183                 .Where(r => r.Active && r.BookID.Equals(request.BookID))
184                 .Count() > 0;
185             bool OverRenewals = _ctx.Checkouts
186                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
187                 .OrderByDescending(c => c.CheckoutDate)
188                 .First()
```

```
189                 .Renewals > 2;
190             if (AlreadyReserved || OverRenewals)
191             {
192                 return Forbid();
193             }
194             DateTime Checkout = _ctx.Checkouts
195                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
196                 .OrderByDescending(c => c.CheckoutDate)
197                 .First()
198                 .CheckoutDate;
199             _ctx.Checkouts
200                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
201                 .OrderByDescending(c => c.CheckoutDate)
202                 .First()
203                 .CheckoutDate = Checkout.AddDays(7);
204             _ctx.Checkouts
205                 .Where(c => c.Active && c.BookID.Equals(request.BookID) &&
                      c.UserID.Equals(userID))
206                 .OrderByDescending(c => c.CheckoutDate)
207                 .First()
208                 .Renewals += 1;
209             _ctx.SaveChanges();
210             return Ok();
211         }
212     }
213 }
214
```