```csharp
using DatabaseConnect;
using DatabaseConnect.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Cryptography.KeyDerivation;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Logging;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Linq;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text;
using static LibraryAppMVC.Models.Models;

namespace LibraryAppMVC.Controllers
{
    [Route("")]
    [ApiExplorerSettings(IgnoreApi = true)]
    public class SwaggerRedirectController : Controller
    {
        [Route("")]
        [HttpGet]
        [ApiExplorerSettings(IgnoreApi = true)]
        public IActionResult RedirectToSwaggerUi()
        {
            return Redirect("swagger");
        }
    }

    [Route("/user/")] // All endpoints checked 2/25/18, logout not working but 
      not important (token dumped client side at logout)
    public class UserController : Controller
    {
        private IConfiguration _config;
        private Context _ctx;
        private readonly ILogger _logger;

        public UserController(IConfiguration config, Context context, 
          ILogger<UserController> logger)
        {
            _config = config;
            _ctx = context;
            _logger = logger;
        }


        [Route("login")]
        [AllowAnonymous]
        [HttpPost]
        public IActionResult CreateToken([FromBody]LoginModel login) // Checked 2/24/18 working
        {
            IActionResult response = Unauthorized();
            var user = Authenticate(login);

            if(user!=null)
            {
                response = BuildToken(user);
            }
            return response;
        }

        [Route("logout")]
        [Authorize]
        [HttpPost]
        public IActionResult Logout() // Checked 2/24/18 NOT working TODO, maybe not important because client dumps token on logout
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;
            _ctx.Users
                .Single(u => u.UserID == userID);
            _ctx.SaveChanges();
            return Ok();
        }

        [Route("info")]
        [Authorize]
        [HttpGet]
        public IActionResult UserInfo()
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            var user = _ctx.Users
                .Single(u => u.SchoolID == schoolID);
            int userID = user.UserID;

            var checkouts = _ctx.Checkouts
                .Where(c => c.Active)
                .Where(c => c.UserID == userID)
                .Include(c => c.Book)
                .ToList();

            var reservations = _ctx.Reservations
                .Where(r => r.Active)
                .Where(r => r.UserID == userID)
                .Include(r => r.Book)
                .ToList();

            foreach(Checkout c in checkouts)
            {
                c.User = null;
            }
            foreach(Reservation r in reservations)
            {
                r.User = null;
            }
            user.PasswordHash = null;
            user.Salt = null;
            var resp = new { checkouts, reservations, user};
            return Json(resp);
        }

        private IActionResult BuildToken(UserModel user)
        {
            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
            var claims = new[]
            {
                new Claim(JwtRegisteredClaimNames.Sub, user.StudentID),
                new Claim(JwtRegisteredClaimNames.Jti, user.TokenVersion.ToString())
            };

            var token = new JwtSecurityToken(
                _config["Jwt:Issuer"],
                _config["Jwt:Issuer"],
                expires: DateTime.Now.AddMinutes(Convert.ToDouble(_config["LoginDurationMinutes"])),
                signingCredentials: creds,
                claims: claims
                );
            return Ok(
                new {
                    token = new JwtSecurityTokenHandler().WriteToken(token),
                    expiration = token.ValidTo
                });
        }

        private UserModel Authenticate(LoginModel login)
        {
            User User;
            UserModel usermodel = null;
            try
            {
                User = _ctx.Users
                    .Single(u => u.SchoolID.Equals(login.Username));
            }
            catch
            {
                return null;  // No user found with specified school ID
            }
            if(VerifyPass(login.Password, User.Salt, User.PasswordHash))
            {
                usermodel = new UserModel { Name = User.FullName, StudentID = User.SchoolID, TokenVersion = User.TokenVersion };
            }
            return usermodel;
        }

        private Boolean VerifyPass(String RawPass, String Salt, String PasswordHash)
        {
            byte[] salt_array = Convert.FromBase64String(Salt);
            String hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
                password: RawPass,
                salt: salt_array,
                prf: KeyDerivationPrf.HMACSHA1,
                iterationCount: 10000,
                numBytesRequested: 256 / 8));
            return hashed.Equals(PasswordHash);
        }
    }


    [Route("/library/")] // All endpoints checked 2/25/18
    public class LibraryController : Controller
    {
        private Context _ctx;

        public LibraryController(Context context)
        {
            _ctx = context;
        }


        [Route("checkout")]
        [HttpPost]
        [Authorize]
        public IActionResult BookCheckout([FromBody]TransactionRequest request) // Checked 2/24/18 working
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;

            if(!_ctx.Books.Any(b => b.BookID == request.BookID))
            {
                return StatusCode(409, "Book does not exist");
            }

            int limit = _ctx.UserUType_rel // Get max checked out books for usertype
                .Include(ut => ut.UType)
                .Single(ut => ut.UserID == userID)
                .UType
                .CheckoutLimit;

            int current = _ctx.Checkouts // Get current user checked out books
                .Where(c => c.Active)
                .Where(c => c.UserID == userID)
                .Count();

            if (current >= limit)  // Check to see if user can checkout more books
            {
                return StatusCode(409, $"You already have checked out {current} books, as many as you can.");
            }

            bool CheckedOut = _ctx.Checkouts
                .Where(c => c.Active && c.BookID.Equals(request.BookID))
                .Count() > 0;

            if (CheckedOut)
            {
                return StatusCode(409, "Already checked out");
            }

            _ctx.Checkouts
                .Add(new Checkout { BookID = request.BookID, UserID = userID, Active=true, CheckoutDate=DateTime.Now, DueDate=DateTime.Now.AddDays(14) });
            _ctx.SaveChanges();
            return Ok();
        }

        [Route("checkin")]
        [HttpPost]
        [Authorize]
        public IActionResult BookCheckin([FromBody]TransactionRequest request) // Checked 2/24/18 working
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;

            if (!_ctx.Books.Any(b => b.BookID == request.BookID))
            {
                return StatusCode(409, "Book does not exist");
            }

            _ctx.Checkouts
                .Where(c => c.BookID == request.BookID && c.UserID == userID)
                .Last()
                .Active = false;
            _ctx.SaveChanges();
            return Ok();
        }

        [Route("reserve")]
        [HttpPost]
        [Authorize]
        public IActionResult ReserveBook([FromBody]TransactionRequest request) // Checked 2/25/18 working
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;

            if (!_ctx.Books.Any(b => b.BookID == request.BookID))
            {
                return StatusCode(409, "Book does not exist");
            }

            Boolean BookAvailable = _ctx.Checkouts
                .Where(c => c.BookID == request.BookID && c.Active)
                .Count() > 0;

            Boolean UserAlreadyReserved = _ctx.Reservations
                .Where(r => r.Active && r.UserID == userID)
                .Count() > 0;

            if(!UserAlreadyReserved || !BookAvailable)
            {
                _ctx.Reservations
                    .Add(new Reservation { BookID = request.BookID, UserID = userID, Datetime = DateTime.Now, Active = true});
                _ctx.SaveChanges();
                return Ok();
            }
            else if(UserAlreadyReserved)
            {
                return StatusCode(409, "You have already reserved this book");
            }
            else if(BookAvailable)
            {
                return StatusCode(409, "This book can be checked out now, not reserved");
            }
            return StatusCode(500);

        }

        [Route("fill_reservation")]
        [HttpPost]
        [Authorize]
        public IActionResult FillReservation([FromBody]TransactionRequest request) // Checked 2/25/18 working
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;

            if (!_ctx.Books.Any(b => b.BookID == request.BookID))
            {
                return StatusCode(409, "Book does not exist");
            }

            Boolean CheckedOut = _ctx.Checkouts
                .Any(c => c.BookID == request.BookID && c.UserID == userID && c.Active == true);

            if(!CheckedOut)
            {
                IActionResult resp = BookCheckout(request);
                _ctx.Reservations
                    .Where(r => r.Active && r.BookID.Equals(request.BookID) && r.UserID.Equals(userID))
                    .OrderByDescending(r => r.Datetime)
                    .First()
                    .Active = false;
                _ctx.SaveChanges();
                return resp;
            }
            else
            {
                return StatusCode(409, "Book already checked out");
            }
        }

        [Route("renew")]
        [HttpPost]
        [Authorize]
        public IActionResult RenewBook([FromBody]TransactionRequest request) // Checked 2/25/18 working
        {
            string schoolID = User.Claims.FirstOrDefault(c => c.Type == ClaimTypes.NameIdentifier).Value;
            int userID = _ctx.Users
                .Single(u => u.SchoolID == schoolID)
                .UserID;

            bool AlreadyReserved = _ctx.Reservations
                .Where(r => r.Active && r.BookID.Equals(request.BookID))
                .Count() > 0;
            bool OverRenewals = _ctx.Checkouts
                .Where(c => c.Active && c.BookID.Equals(request.BookID) && c.UserID.Equals(userID))
                .OrderByDescending(c => c.CheckoutDate)
                .First()
                .Renewals > 2;
            if(AlreadyReserved || OverRenewals)
            {
                return Forbid();
            }
            DateTime Checkout = _ctx.Checkouts
                .Where(c => c.Active && c.BookID.Equals(request.BookID) && c.UserID.Equals(userID))
                .OrderByDescending(c => c.CheckoutDate)
                .First()
                .CheckoutDate;
            _ctx.Checkouts
                .Where(c => c.Active && c.BookID.Equals(request.BookID) && c.UserID.Equals(userID))
                .OrderByDescending(c => c.CheckoutDate)
                .First()
                .CheckoutDate = Checkout.AddDays(7);
            _ctx.Checkouts
                .Where(c => c.Active && c.BookID.Equals(request.BookID) && c.UserID.Equals(userID))
                .OrderByDescending(c => c.CheckoutDate)
                .First()
                .Renewals += 1;
            _ctx.SaveChanges();
            return Ok();
        }
    }


    [Route("/simple/")] // All endpoints checked 2/25/18
    public class SimpleController : Controller
    {
        private Context _ctx;

        public SimpleController(Context context)
        {
            _ctx = context;
        }

        [AllowAnonymous]
        [Route("books")]
        [HttpGet]
        public IActionResult GetABook(string title, int page = 1) // Checked
          2/25/18 working
        {
            List<Book> a;
            if (title != null)  // Title specified
            {
                a = _ctx.Books
                    .Where(b => b.Title.Contains(title))
                    .Include(book => book.Cover)
                    .Include(book => book.AuthorBooks)
                        .ThenInclude(ab => ab.Author)
                    .ToList();
            }
            else  // Title not specified
            {
                if (page < 1) { page = 1; }
                int pos_i = (page - 1) * 10;
                int pos_f = page * 10;
                int count = _ctx.Books.Count();
                if (pos_f > count) { pos_f = count; }
                if (pos_i > count) { a = new List<Book>(); }
                else
                {
                    a = _ctx.Books
                        .Include(book => book.Cover)
                        .Include(book => book.AuthorBooks)
                            .ThenInclude(ab => ab.Author)
                        .ToList()
                        .GetRange(pos_i, (pos_f - pos_i));
                }
            }

            foreach(Book b in a)
            {
                List<String> AuthorList = new List<String>();
                //b.Cover.Books = null;
                foreach(AuthorBook ab in b.AuthorBooks)
                {
                    AuthorList.Add(ab.Author.Name);
                }
                b.Authors = AuthorList;
                b.AuthorBooks = null;
            }
            return Json(a);
        }

        [Route("checkouts")]
        [AllowAnonymous]
        [HttpGet]
        public IActionResult GetCheckouts() // Checked 2/25/18 working
        {
            var CheckoutList = _ctx.Checkouts
                .Include(c => c.Book)
                .Where(c => c.Active)
                .ToList();
            return Json(CheckoutList);
        }

        [Route("reservations")]
        [AllowAnonymous]
        [HttpGet]
        public IActionResult GetReservations() // Checked 2/25/18 working
        {
            var CheckoutList = _ctx.Reservations
                .Include(r => r.Book)
                .Where(r => r.Active)
                .ToList();
            return Json(CheckoutList);
        }
    }


    [Route("/dev/")] // All endpoints checked 2/25/18
    public class DevController : Controller
    {
        private Context _ctx;
        public DevController(Context context)
        {
            _ctx = context;
        }

        [Route("adduser")]
        [HttpPost]
        public IActionResult AddUser([FromBody]NewUser newuser) // Checked 
          2/25/18 working
        {
            if (newuser.UserTypeInt == 0) { newuser.UserTypeInt = 1; }
            User user = new User() { SchoolID = newuser.Username, Password = 
              newuser.Password };
            byte[] salt = new byte[128 / 8];
            using (var rng = RandomNumberGenerator.Create())
            {
                rng.GetBytes(salt);
            }
            string hashed = Convert.ToBase64String(KeyDerivation.Pbkdf2(
                    password: user.Password,
                    salt: salt,
                    prf: KeyDerivationPrf.HMACSHA1,
                    iterationCount: 10000,
                    numBytesRequested: 256 / 8));
            user.Salt = Convert.ToBase64String(salt);
            user.PasswordHash = hashed;
            _ctx.Users.Add(user);
            _ctx.SaveChanges();
            int UserID = _ctx.Users
                .Single(u => u.SchoolID == user.SchoolID)
                .UserID;
            _ctx.UserUType_rel
                .Add(new UserUType { UserID = UserID, UTypeID = 1 });
            _ctx.SaveChanges();
            return Ok();
        }
    }
}

```