Source code

Frontend

Navbar.jsx

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import './Navbar.css';
import 'bootstrap/dist/css/bootstrap.css';
import { Button, Navbar, Nav } from 'react-bootstrap';
import mainLogo from './ALogo.png';
import LoginModal from '../Login/login';
import SignInModal from '../Login/signup';
import { connect } from 'react-redux';
import { logoutUser } from '../../redux/userAction';

const Navigation = ({ isAuthenticated, userName, logoutUser }) => {
  const [loginModalShow, setLoginModalShow] = useState(false);
  const [registerModalShow, setRegisterModalShow] = useState(false);

  const handleLoginClick = () => {
    setLoginModalShow(true);
  };

  const handleLoginModalClose = () => {
    setLoginModalShow(false);
  };

  const handleSignUpClick = () => {
    setRegisterModalShow(true);
  };

  const handleRegisterModalClose = () => {
    setRegisterModalShow(false);
  };

  const handleLogout = () => {
    // Dispatch the logout action
    logoutUser();
  };
  return (
    <div>
      <style>
        @import
url('https://fonts.googleapis.com/css2?family=Courgette&display=swap');
      </style>
      <Navbar className="navbar navbar-expand-lg navbar-inner">
        <Link to="/" className="navbar-brand">
          <img className="logo" src={mainLogo} alt="Healthcare Icon" />
          <span className='logoName'>ABC Healthcare</span>
        </Link>
        <Nav className="navoptions">
            <Link to="/cart">
              <Button className="navbutton">Cart</Button>
            </Link>
          {isAuthenticated ? (
            <>
              <h4 id="username">{userName}</h4>
              <Button variant="primary" onClick={handleLogout}>
                Log Out
              </Button>
            </>
          ) : (
            <>
              <Button className='navbutton' variant="outline-primary"
onClick={handleSignUpClick}>
                Sign Up
              </Button>
              <Button className='navbutton' variant="primary"
onClick={handleLoginClick}>
                Sign In
              </Button>
            </>
          )}
        </Nav>
      </Navbar>

      {/* Render the login modal */}
      <LoginModal show={loginModalShow}
handleClose={handleLoginModalClose} />
      {/* Render the register modal */}
      <SignInModal show={registerModalShow}
handleClose={handleRegisterModalClose} />
    </div>
  );
};

const mapStateToProps = (state) => ({
  isAuthenticated: state.user.isAuthenticated,
  userName: state.user.userName, // Access the userName from the Redux
state
});

const mapDispatchToProps = {
  logoutUser,
};

export default connect(mapStateToProps, mapDispatchToProps)(Navigation);
```

CartPage.jsx

```
import React, { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { getCartItems, addToCart, deleteCartItem } from
'../../redux/cartAction';
import './cartPage.css'

const CartPage = ({ clearCart, removeFromCart }) => {

    const dispatch = useDispatch();
    const cartItems = useSelector(state => state.cart.cartItems); // Make
sure you use the correct selector

    useEffect(() => {
      // Dispatch the action when the component mounts to fetch cart items
      dispatch(getCartItems());
    }, [dispatch]);
```

```
  return (
    <div className="container cartpage">
      <h2 className="text-center carttitle">Cart</h2>
      <table className="table">
        <thead>
          <tr>
            <th>Product</th>
            <th>Name</th>
            <th>Quantity</th>
            <th>Price</th>
            <th>Total</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          {cartItems.map((item) => (
            <tr key={item.id}>
              <td>
                <img src={item.medItems.imageUrl} alt={item.name}
className="product-image" />
              </td>
              <td>{item.medItems.itemName}</td>
              <td>{item.quantity}</td>
              <td>${item.medItems.price}</td>
              <td>${item.quantity * item.medItems.price}</td>
              <td>
                <button
                  className="btn btn-danger"
                  onClick={() => removeFromCart(item.id)}
                >
                  Remove
                </button>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
      <div className="cart-summary">
        <p>Total Cost: ${cartItems.reduce((total, item) => total +
item.price * item.quantity, 0)}</p>
        <button className="btn btn-danger" onClick={clearCart}>
          Clear Cart
        </button>
        <button className="btn btn-primary">
          Confirm
        </button>
      </div>
    </div>
  );
};

export default CartPage;
```

AdminPanel.jsx

```
import React, { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { getCategories, addCategory, deleteCategory, getItems, addItem,
deleteItem } from '../../redux/adminAction';
import './adminstyle.css';

const AdminPanel = () => {
  const categories = useSelector(state => state.admin.categories);
  const items = useSelector(state => state.admin.items);
  const dispatch = useDispatch();

  const [newCategory, setNewCategory] = useState('');
  const [showCategories, setShowCategories] = useState(false);
  const [showItems, setShowItems] = useState(false);
  const [newItem, setNewItem] = useState({
    itemName: '',
    categoryId: '',
    description: '',
    imageUrl: '',
    price: '',
    seller: '',
    medCategory: {
      categoryId: 0, // Set the categoryId to 0 or the appropriate value
      categoryName: "string", // Set the categoryName to the appropriate
value
    },
  });

  useEffect(() => {
    dispatch(getCategories());
    dispatch(getItems());
  }, [dispatch]);


console.log('Categories:', categories); // Log categories if they exist
console.log('Items:', items); // Log items if they exist


//category add
  const handleAddCategory = () => {
    dispatch(addCategory({ categoryName: newCategory }));
    setNewCategory('');
    console.log("new categorn adding",newCategory);
  };

//category delete

const handleDeleteCategory = (categoryId) => {
  dispatch(deleteCategory(categoryId))
    .then(() => {
      console.log("deleting category",categoryId)
      dispatch(getCategories());
    })
    .catch((error) => {
      // Handle any errors that occur during deletion or fetching.
      console.error("Error deleting category", error);
    });
};


//items add
```

```
  const handleAddItem = () => {
    dispatch(addItem(newItem));
    console.log(newItem);
    // Reset the newItem fields
    setNewItem({
      itemName: '',
      categoryId: '',
      description: '',
      imageUrl: '',
      price: '',
      seller: '',
    });
  };


//deletem item
  const handleDeleteItem = (itemId) => {
    dispatch(deleteItem(itemId))
    .then(() => {
      console.log("deleting item",itemId)
      dispatch(getItems());
    })
    .catch((error) => {
      // Handle any errors that occur during deletion or fetching.
      console.error("Error deleting item", error);
    });
};

  return (
    <div className='adminbox'>
      <button className='manage' onClick={() =>
setShowCategories(!showCategories)}>
        Manage Categories
      </button>

      {showCategories && (
        <div>
          {/* <h2>Categories</h2> */}
          {categories && categories.length > 0 ? (
            <table>
              <thead>
                <tr>
                  <th>Category Id</th>
                  <th>Category Name</th>
                  <th>Action</th>
                </tr>
              </thead>
              <tbody>
              {categories.map((category) => (
                <tr key={category.categoryId}>
                  <td>{category.categoryId}</td>
                  <td>{category.categoryName}</td>
                  <td><button onClick={() =>
handleDeleteCategory(category.categoryId)}>Delete</button></td>
                </tr>
              ))}

              </tbody>
            </table>
          ) : (
            <p>Loading categories...</p>
          )}
          <div>
            <input
              type="text"
              placeholder="Enter Category Name"
              value={newCategory}
              onChange={(e) => setNewCategory(e.target.value)}
            />
            <button id='addCategoryBtn' onClick={handleAddCategory}>Add
Category</button>
          </div>
        </div>
      )}

      <button className='manage' onClick={() =>
setShowItems(!showItems)}>Manage Items</button>

      {showItems && (
        <div>
          {/* <h2>Items</h2> */}
          {items && items.length > 0 ? (
            <table>
              <thead>
                <tr>
                  <th>Id</th>
                  <th>Item Name</th>
                  <th>Category ID</th>
                  <th>Description</th>
                  <th>Image Url</th>
                  <th>Price</th>
                  <th>Seller</th>
                  <th>Action</th>
                </tr>
              </thead>
              <tbody>
              {items.map((item) => (
                <tr key={item.itemId}>
                  <td>{item.itemId}</td>
                  <td>{item.itemName}</td>
                  <td>{item.categoryId}</td>
                  <td className='longText
description'>{item.description}</td>
                  <td className='longText imageUrl'>{item.imageUrl}</td>
                  <td>{item.price}</td>
                  <td>{item.seller}</td>
                  <td><button onClick={() =>
handleDeleteItem(item.itemId)}>X</button></td>
                </tr>
              ))}

              </tbody>
            </table>
          ) : (
            <p>Loading items...</p>
          )}
          <div className='entryarea'>
          <input
              type="text"
              placeholder="Item Name"
              value={newItem.itemName}
              onChange={(e) => setNewItem({ ...newItem, itemName:
e.target.value })}
            />
            <input
              type="text"
              placeholder="Category ID"
              value={newItem.categoryId}
              onChange={(e) => setNewItem({ ...newItem,
categoryId:Number(e.target.value) })}
            />
            <input
              type="text"
              placeholder="Description"
              value={newItem.description}
              onChange={(e) => setNewItem({ ...newItem, description:
e.target.value })}
            />
            <input
              type="text"
              placeholder="Image URL"
              value={newItem.imageUrl}
              onChange={(e) => setNewItem({ ...newItem, imageUrl:
e.target.value })}
            />
            <input
              type="text"
              placeholder="Price"
              value={newItem.price}
              onChange={(e) => setNewItem({ ...newItem, price:
parseFloat(e.target.value) })}
            />
            <input
              type="text"
              placeholder="Seller"
              value={newItem.seller}
              onChange={(e) => setNewItem({ ...newItem, seller:
e.target.value })}
            />
            <button onClick={handleAddItem}>Add Item</button>
          </div>
        </div>
      )}
    </div>
  );
};

export default AdminPanel;
```

Cards.jsx

```
import React, { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import './card.css'; // Import the CSS file for styling
import fetchMedItems from '../../redux/taskAction';
import { addToCart } from '../../redux/cartAction';

const Card = () => {
  const dispatch = useDispatch();
  const medItems = useSelector(state => state.task.medItems);

  const [newItem, setNewItem] = useState({
    userId: 1,
    itemId:null,
    quantity: 1,
    medItems: {
      itemId: 0,
      categoryId: 0,
      itemName: "string",
      price: 0,
      imageUrl: "string",
      seller: "string",
      description: "string",
      medCategory: {
        categoryId: 0,
        categoryName: "string"
      }
    }
  });

  useEffect(() => {
    dispatch(fetchMedItems());
  }, [dispatch]);


  const handleAddCart = (item) => {
    if (item.itemId === null) {
      alert('Please select an item to add to the cart.');
      return;
    }

    setNewItem({ ...newItem, itemId: item.itemId });
    console.log(newItem);
    dispatch(addToCart(newItem)); // Dispatch the addToCart action
    console.log('Item dispatched to the cart.');
  };

  if (!medItems) {
    return <div>Loading...</div>;
  }

  return (

    <div className="card-container">
       {console.log('medItems:', medItems)}
      {medItems.map((item) => (

        <div key={item.itemId} className={`card ${item.Name}`}>
          <div className={`$cardImage {item.ItemId}`}>
            <img src={`${item.imageUrl}`} alt={`${item.Name}`} style={{
width: '150px', display: 'block', margin: '0 auto' }}></img>
          </div>
          <div className="card-title">
            <h3>{item.itemName}</h3>
          </div>
          <div className="card-description">
            <h6>{item.description}</h6>
          </div>
          <p className="card-text">{item.Price}</p>
          <div className="form-group d-flex align-items-center">
            <button
              className="btn btn-primary add-to-cart-button ml-auto"
              onClick={() => handleAddCart(item)}
            >
              Add to Cart
            </button>
          </div>
        </div>
      ))}
    </div>
  );
};

export default Card;
```

AdminAction

```
import axios from 'axios';
import {GET_CATEGORIES, ADD_CATEGORY, DELETE_CATEGORY, GET_ITEMS,
ADD_ITEM, DELETE_ITEM,} from './adminActionType'


export const getCategories = () => async (dispatch) => {
  try {
    const response = await
axios.get('https://ehealthcareappapi.azurewebsites.net/api/MedCategories')
; // Replace with your API endpoint
    dispatch({ type: GET_CATEGORIES, payload: response.data });
    console.log(response.data);
  } catch (error) {
    console.log('no data received')
  }
};

export const addCategory = (categoryData) => async (dispatch) => {
  try {
    console.log(categoryData);
    const response = await
axios.post('https://ehealthcareappapi.azurewebsites.net/api/MedCategories'
, categoryData); // Replace with your API endpoint
    dispatch({ type: ADD_CATEGORY, payload: response.data });
    console.log(response);
  } catch (error) {
    // Handle error
  }
};

export const deleteCategory = (categoryId) => async (dispatch) => {
  try {
    await
axios.delete(`https://ehealthcareappapi.azurewebsites.net/api/MedCategorie
s/${categoryId}`); // Replace with your API endpoint
    dispatch({ type: DELETE_CATEGORY, payload: categoryId });
  } catch (error) {

  }
};


export const getItems = () => async (dispatch) => {
  try {
    const response = await
axios.get('https://ehealthcareappapi.azurewebsites.net/api/MedItems');
    dispatch({ type: GET_ITEMS, payload: response.data }); // Use
GET_ITEMS here
    console.log(response.data);
  } catch (error) {
    // Handle error
  }
};

export const addItem = (itemData) => async (dispatch) => {
  try {
    const response = await
axios.post('https://ehealthcareappapi.azurewebsites.net/api/MedItems',
itemData);
```

```
    dispatch({ type: ADD_ITEM, payload: response.data }); // Use ADD_ITEM
here
  } catch (error) {
    // Handle error
  }
};

export const deleteItem = (itemId) => async (dispatch) => {
  try {
    await
axios.delete(`https://ehealthcareappapi.azurewebsites.net/api/MedItems/${i
temId}`);
    dispatch({ type: DELETE_ITEM, payload: itemId }); // Use DELETE_ITEM
here
  } catch (error) {
    // Handle error
  }

};
```

AdminReducer

```
import {GET_CATEGORIES, ADD_CATEGORY, DELETE_CATEGORY, GET_ITEMS,
ADD_ITEM, DELETE_ITEM,} from './adminActionType'
```

```



const initialState = {
  categories: [],
  items: [],
};



  const adminReducer = (state = initialState, action) => {
    switch (action.type) {
      case GET_CATEGORIES:
        return { ...state, categories: action.payload };
      case ADD_CATEGORY:
        return { ...state, categories: [...state.categories,
action.payload] };
      case DELETE_CATEGORY:
        return {
          ...state,
          categories: state.categories.filter((category) =>
category.CategoryId !== action.payload),
        };

      // Cases for items
      case GET_ITEMS:
        return { ...state, items: action.payload };
      case ADD_ITEM:
        return { ...state, items: [...state.items, action.payload] };
      case DELETE_ITEM:
        return {
          ...state,
          items: state.items.filter((item) => item.ItemId !==
action.payload),
        };
      default:
        return state;
    }
  };


  export default adminReducer;
```

CartAction

```
import axios from 'axios';
import {GET_CART, ADD_CART, DELETE_CART,} from './cartActionType'

export const getCartItems = () => async (dispatch) => {
  try {
    const response = await
axios.get('https://ehealthcareappapi.azurewebsites.net/api/MedCarts');
    dispatch({ type: GET_CART, payload: response.data }); // Use GET_CART
here
    console.log("recieved file",response.data);
  } catch (error) {
    console.error('Error fetching cart items:', error);
    // Handle error
  }
};


export const addToCart = (itemData) => async (dispatch) => {
    try {
      const response = await
axios.post('https://ehealthcareappapi.azurewebsites.net/api/MedCarts',
itemData);
      dispatch({ type: ADD_CART, payload: response.data });
      console.log("cartActionadding",itemData);
    } catch (error) {
      // Handle error
    }
  };


export const deleteCartItem = (itemId) => async (dispatch) => {
  try {
    await
axios.delete(`https://ehealthcareappapi.azurewebsites.net/api/MedCarts/${i
temId}`);
    dispatch({ type: DELETE_CART, payload: itemId }); // Use DELETE_CART
here
  } catch (error) {
    // Handle error
  }

};
```

CartReducer

```
import {GET_CART, ADD_CART, DELETE_CART} from './cartActionType'

const initialState = {
  cartItems: [],
};
```

```
  const cartReducer = (state = initialState, action) => {
    switch (action.type) {
      case GET_CART:
        return { ...state, cartItems: action.payload };
      case ADD_CART:
        return { ...state, cartItems: [...state.cartItems, action.payload]
};
      case DELETE_CART:
        return {
          ...state,
          cartItems: state.cartItems.filter((item) => item.ItemId !==
action.payload),
        };
      default:
        return state;
    }
  };


  export default cartReducer;
```

Task Action

```
import axios from 'axios';
import { FETCH_MED_ITEMS_SUCCESS } from "./taskActionTypes";
// Rest of your action creator code

export const fetchMedItemsSuccess = (medItems) => ({
  type: FETCH_MED_ITEMS_SUCCESS,
  payload: medItems,
});

export const fetchMedItems = () => {
  return (dispatch) => {
    // Update the API URL according to your actual API endpoint
    axios.get('https://ehealthcareappapi.azurewebsites.net/api/MedItems')
      .then((response) => {
        dispatch(fetchMedItemsSuccess(response.data));
        console.log('meds received', response.data)
      })
      .catch((error) => {
        console.error('Error fetching med items:', error);
      });
  };
};


export default fetchMedItems;
```

Task reducer

```
import * as types from "./taskActionTypes";

const initialState = {
  medItems: [],
};

const taskReducer = (state = initialState, action) => {
  switch (action.type) {
    case types.FETCH_MED_ITEMS_SUCCESS:
      return {
         ...state,
         medItems: action.payload,
         error: null,
        };
    default:
      return state;
  }
};

export default taskReducer;
```

UserAction

```
import axios from 'axios';
import { REGISTER_USER, LOGIN_USER, LOGOUT_USER } from './userActionType';
```

```
export const registerUser = (userData) => {
  return async (dispatch) => {
    try {
      const response = await
axios.post('https://ehealthcareappapi.azurewebsites.net/api/UserControls',
userData);
      console.log(userData);
      console.log(response);

      if (response.status === 201) {
        // Registration is successful
        dispatch({ type: REGISTER_USER, payload: response.data });
        return response.data; // Return the response data
      }

      // Handle other success codes or messages here
      return { error: 'User registration error: ' + response.data };
    } catch (error) {
      if (error.response && error.response.status === 409) {
        return { error: 'Username already exists. Please choose a
different username.' };
      }

      if (error.response && error.response.status === 400) {
        // HTTP status code 400 indicates a bad request
        return { error: 'Bad request. Please check your registration
data.' };
      }

      // Handle other registration errors
      console.error('Registration error:', error);
      return { error: 'Registration failed. Please try again.' };
    }
  };
};

export const loginUser = (userData) => {
  return async (dispatch) => {
    try {
      // Make an API request to log in the user
      const response = await
axios.post('https://ehealthcareappapi.azurewebsites.net/api/UserControls/l
ogin', userData);
      dispatch({ type: LOGIN_USER, payload: response.data });
      console.log(userData);
    } catch (error) {
      // Handle login error
    }
  };
};

export const logoutUser = () => {
  return { type: LOGOUT_USER };
};
```

User Reducer

```
import { REGISTER_USER, LOGIN_USER, LOGOUT_USER } from './userActionType';

const initialState = {
    user: null,
    isAuthenticated: false,
};

const userReducer = (state = initialState, action) => {
  switch (action.type) {
    case REGISTER_USER:
      return {
        ...state,
        user: action.payload,
        isAuthenticated: true,
        // You can update other user-related state properties here if
needed
      };
    case LOGIN_USER:
      return {
        ...state,
        user: action.payload,
        isAuthenticated: true,
        userName: action.payload.userName,
        // You can update other user-related state properties here if
needed
      };
    case LOGOUT_USER:
      return {
        ...state,
        user: null,
        isAuthenticated: false,
        // You can reset other user-related state properties here if
needed
      };
    default:
      return state;
  }
};

export default userReducer;
```

App.js

```
import React from 'react';
import {BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar/Navbar';
import 'bootstrap/dist/css/bootstrap.css';
import Card from './components/Cards/card';
import CartPage from './components/Cart/cartPage';
import AdminPanel from './components/AdminPanel/adminPanel';




function App() {
  return (
    <Router>
      <div>
        <Navbar/>
        <Routes>
```

```
          <Route path='/' element={<Card />} />
          <Route path='/Admin' element={<AdminPanel />} />
          <Route path='/Cart' element={<CartPage />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

Backend

Medcarts Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using healthcareBackend_.NET.Data;
using healthcareBackend_.NET.Models;

namespace healthcareBackend_.NET.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MedCartsController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public MedCartsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/MedCarts
        [HttpGet]
        public async Task<ActionResult<IEnumerable<MedCart>>> GetMedCart()
        {
                        var medCarts = await _context.MedCart
              .Include(mc => mc.MedItems) // Include the MedItems navigation property
              .ToListAsync();

                        if (_context.MedCart == null)
          {
              return NotFound();
          }
            return await _context.MedCart.ToListAsync();
        }

                // GET: api/MedCarts/5
                [HttpGet("{id}")]
                public async Task<ActionResult<MedCart>> GetMedCart(int id)
                {
                        var medCart = await _context.MedCart
                                .Include(mc => mc.MedItems) // Include the MedItems navigation
property
                                .FirstOrDefaultAsync(mc => mc.CartId == id);

                        if (medCart == null)
                        {
                                return NotFound();
                        }

                        return medCart;
                }

                // PUT: api/MedCarts/5
                // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
                [HttpPut("{id}")]
        public async Task<IActionResult> PutMedCart(int id, MedCart medCart)
        {
            if (id != medCart.CartId)
            {
                return BadRequest();
            }
            _context.Entry(medCart).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MedCartExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }


        // POST: api/MedCarts
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]

                public async Task<ActionResult<MedCart>> PostMedCart(MedCart medCart)
                {
                        // Fetch the MedItem based on itemId
                        var medItem = await _context.MedItems.FindAsync(medCart.ItemId);

                        if (medItem == null)
                        {
                                return NotFound("MedItem not found");
                        }

                        // Fetch the MedCategory based on categoryId in the MedItem
                        var medCategory = await
_context.MedCategory.FindAsync(medItem.CategoryId);

                        if (medCategory == null)
                        {
                                return NotFound("MedCategory not found");
                                }

                                // Set the MedItems and MedCategory properties
                                medCart.MedItems = medItem;
                                medCart.MedItems.MedCategory = medCategory;

                                // Add the MedCart to the context
                                _context.MedCart.Add(medCart);
                                await _context.SaveChangesAsync();

                                return CreatedAtAction("GetMedCart", new { id = medCart.CartId },
medCart);
                        }

                        // DELETE: api/MedCarts/5
                        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteMedCart(int id)
        {
            if (_context.MedCart == null)
            {
                return NotFound();
            }
            var medCart = await _context.MedCart.FindAsync(id);
            if (medCart == null)
            {
                return NotFound();
            }

            _context.MedCart.Remove(medCart);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool MedCartExists(int id)
        {
            return (_context.MedCart?.Any(e => e.CartId == id)).GetValueOrDefault();
        }
    }
}
```

MedCategories Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using healthcareBackend_.NET.Data;
using healthcareBackend_.NET.Models;

namespace healthcareBackend_.NET.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MedCategoriesController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public MedCategoriesController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/MedCategories
        [HttpGet]
        public async Task<ActionResult<IEnumerable<MedCategory>>> GetMedCategory()
        {
          if (_context.MedCategory == null)
          {
              return NotFound();
          }
            return await _context.MedCategory.ToListAsync();
        }

        // GET: api/MedCategories/5
        [HttpGet("{id}")]
        public async Task<ActionResult<MedCategory>> GetMedCategory(int id)
        {
          if (_context.MedCategory == null)
          {
              return NotFound();
          }
            var medCategory = await _context.MedCategory.FindAsync(id);
            if (medCategory == null)
            {
                return NotFound();
            }

            return medCategory;
        }

        // PUT: api/MedCategories/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult> PutMedCategory(int id, MedCategory medCategory)
        {
            if (id != medCategory.CategoryId)
            {
                return BadRequest();
            }

            _context.Entry(medCategory).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MedCategoryExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/MedCategories
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<MedCategory>> PostMedCategory(MedCategory
medCategory)
```

```
        {
          if (_context.MedCategory == null)
          {
              return Problem("Entity set 'ApplicationDbContext.MedCategory'  is null.");
          }
            _context.MedCategory.Add(medCategory);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetMedCategory", new { id = medCategory.CategoryId },
medCategory);
        }

        // DELETE: api/MedCategories/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteMedCategory(int id)
        {
            if (_context.MedCategory == null)
            {
                return NotFound();
            }
            var medCategory = await _context.MedCategory.FindAsync(id);
            if (medCategory == null)
            {
                return NotFound();
            }

            _context.MedCategory.Remove(medCategory);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool MedCategoryExists(int id)
        {
            return (_context.MedCategory?.Any(e => e.CategoryId == id)).GetValueOrDefault();
        }
    }
}
```

MedItems COntroller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using healthcareBackend_.NET.Data;
using healthcareBackend_.NET.Models;

namespace healthcareBackend_.NET.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MedItemsController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public MedItemsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/MedItems
        [HttpGet]
        public async Task<ActionResult<IEnumerable<MedItems>>> GetMedItems()
        {
          if (_context.MedItems == null)
          {
              return NotFound();
          }
            return await _context.MedItems.ToListAsync();
        }

        // GET: api/MedItems/5
        [HttpGet("{id}")]
        public async Task<ActionResult<MedItems>> GetMedItems(int id)
        {
          if (_context.MedItems == null)
          {
              return NotFound();
          }
            var medItems = await _context.MedItems.FindAsync(id);

            if (medItems == null)
            {
                return NotFound();
            }
            return medItems;
        }

        // PUT: api/MedItems/5
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPut("{id}")]
        public async Task<IActionResult> PutMedItems(int id, MedItems medItems)
        {
            if (id != medItems.ItemId)
            {
                return BadRequest();
            }

            _context.Entry(medItems).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MedItemsExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/MedItems
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<MedItems>> PostMedItems(MedItems medItems)
                        //{
                        //  if (_context.MedItems == null)
                        //  {
                        //      return Problem("Entity set 'ApplicationDbContext.MedItems'  is null.");
                        //  }
                        //    _context.MedItems.Add(medItems);
                        //    await _context.SaveChangesAsync();

                        //    return CreatedAtAction("GetMedItems", new { id = medItems.ItemId },
medItems);
                        //}
                        {
                            // Check if the category exists before creating the item
                            var existingCategory = await
_context.MedCategory.FindAsync(medItems.CategoryId);
                            if (existingCategory == null)
                            {
                                // Return a bad request response because the category doesn't
exist
                                return BadRequest("The specified category doesn't exist.");
                            }

                            // Link the item to the existing category
                            medItems.MedCategory = existingCategory;

                            // Add the item to the context and save changes
                            _context.MedItems.Add(medItems);
                            await _context.SaveChangesAsync();

                            return CreatedAtAction("GetMedItems", new { id = medItems.ItemId },
medItems);
                        }



                        // DELETE: api/MedItems/5
                        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteMedItems(int id)
        {
            if (_context.MedItems == null)
            {
                return NotFound();
            }
            var medItems = await _context.MedItems.FindAsync(id);
            if (medItems == null)
            {
                return NotFound();
            }
```

```
            _context.MedItems.Remove(medItems);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        private bool MedItemsExists(int id)
        {
            return (_context.MedItems?.Any(e => e.ItemId == id)).GetValueOrDefault();
        }
    }
}
```

User Controller Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using healthcareBackend_.NET.Data;
using healthcareBackend_.NET.Models;

namespace healthcareBackend_.NET.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserControlsController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public UserControlsController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: api/UserControls
        [HttpGet]
        public async Task<ActionResult<IEnumerable<UserControl>>> GetUserControl()
        {
          if (_context.UserControl == null)
          {
            return NotFound();
          }
            return await _context.UserControl.ToListAsync();
        }

        // GET: api/UserControls/5
        [HttpGet("{id}")]
        public async Task<ActionResult<UserControl>> GetUserControl(int id)
        {
          if (_context.UserControl == null)
          {
              return NotFound();
          }
            var userControl = await _context.UserControl.FindAsync(id);

            if (userControl == null)
            {
                return NotFound();
            }

            return userControl;
        }


                        // PUT: api/UserControls/5
                        // To protect from overposting attacks, see
https://go.microsoft.com/fwlink/?linkid=2123754
                        [HttpPut("{id}")]
        public async Task<IActionResult> PutUserControl(int id, UserControl userControl)
        {
            if (id != userControl.UserId)
            {
                return BadRequest();
            }

            _context.Entry(userControl).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!UserControlExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
        }

        // POST: api/UserControls
        // To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754
        [HttpPost]
        public async Task<ActionResult<UserControl>> PostUserControl(UserControl userControl)
        {
            if (_context.UserControl == null)
            {
                return Problem("Entity set 'ApplicationDbContext.UserControl'  is null.");
            }


            var usernameExists = await CheckUsernameExists(userControl.UserName);

            if (usernameExists)
            {
                        return Conflict("Username already exists. Please choose a different username.");
            }



                                _context.UserControl.Add(userControl);
            await _context.SaveChangesAsync();

            return CreatedAtAction("GetUserControl", new { id = userControl.UserId }, userControl);



        }

        // DELETE: api/UserControls/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteUserControl(int id)
        {
            if (_context.UserControl == null)
```

```
    {
        return NotFound();
    }
    var userControl = await _context.UserControl.FindAsync(id);
    if (userControl == null)
    {
        return NotFound();
    }

    _context.UserControl.Remove(userControl);
    await _context.SaveChangesAsync();

    return NoContent();
}




// Authentication endpoint
[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] LoginRequest loginRequest)
{
        var user = await _context.UserControl
                .SingleOrDefaultAsync(u => u.UserName == loginRequest.Username);

        if (user == null)
        {
                return NotFound("User not found");
        }

        if (user.Password == loginRequest.Password)
        {
                // Password matches; user is authenticated
                // You can return a token or other authentication response here
                return Ok("Authentication successful");
        }
        else
        {
                // Password does not match
                return Unauthorized("Authentication failed");
        }
}

// GETapi/UserControls/registration
```

```
        [HttpGet("check-username/{username}")]
        private async Task<bool> CheckUsernameExists(string username)
        {
                var existingUser = await _context.UserControl.FirstOrDefaultAsync(u =>
u.UserName == username);
                return existingUser != null;
        }




                private bool UserControlExists(int id)
        {
            return (_context.UserControl?.Any(e => e.UserId == id)).GetValueOrDefault();
        }
    }
}
```

ApplicationDb COntext

```
using healthcareBackend_.NET.Models;
using Microsoft.EntityFrameworkCore;

namespace healthcareBackend_.NET.Data
{
        public class ApplicationDbContext : DbContext
        {
                public DbSet<MedItems> MedItems { get; set; }
                public DbSet<MedCart> MedCart { get; set; }

                public DbSet<UserControl> UserControl { get; set; }

                public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
: base(options) { }

                protected override void OnModelCreating(ModelBuilder modelBuilder)
                {

                        modelBuilder.Entity<MedItems>()
                                .HasOne(item => item.MedCategory)
                                .WithMany()
                                .HasForeignKey(item => item.CategoryId);
```

```
                    modelBuilder.Entity<MedCart>()
                        .HasOne(cart => cart.MedItems)
                        .WithMany()
                        .HasForeignKey(cart => cart.ItemId);
                }

                public DbSet<healthcareBackend_.NET.Models.MedCategory>? MedCategory {
get; set; }
        }
}
```

Login Request

```
using System.ComponentModel.DataAnnotations;

namespace healthcareBackend_.NET.Models
{
        public class LoginRequest
        {
                [Required]
                public string Username { get; set; }

                [Required]
                public string Password { get; set; }
        }
}
```

MedCArt

```
using System.ComponentModel.DataAnnotations;

namespace healthcareBackend_.NET.Models
{
        public class MedCart
        {
                [Key]
                public int CartId { get; set; }
                public int UserId { get; set; } // Foreign key to User table
                public int ItemId { get; set; } // Foreign key to Item table
                public int Quantity { get; set; }

                public virtual MedItems MedItems { get; set; }
        }
}
```

MedCategory

```
using System.ComponentModel.DataAnnotations;

namespace healthcareBackend_.NET.Models
{
	public class MedCategory
	{
		[Key]
		public int CategoryId { get; set; }
		public string CategoryName { get; set; }
	}
}
```

MedItems

```
using System.ComponentModel.DataAnnotations;

namespace healthcareBackend_.NET.Models
{
	public class MedItems
	{
		[Key]
		public int ItemId { get; set; }
		public int CategoryId { get; set; } // Foreign key to Category table
		public string ItemName { get; set; }
		public decimal Price { get; set; }
		public string ImageUrl { get; set; }
		public string Seller { get; set; }
		public string Description { get; set; }


		//referencing the medcategory to use in item
		public virtual MedCategory MedCategory{ get; set; }
	}
}
```

UserControl

```
using System.ComponentModel.DataAnnotations;

namespace healthcareBackend_.NET.Models
{
	public class UserControl
	{
        [Key]
        public int UserId { get; set; }
        public string UserName { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public string Access { get; set; }
    }
}
```