React Routes

Your assignment is to continue last weeks assignment, by adding routes and a login-prototype. You will install React Router version 6.4 and configure your React app to use routes and links. You will add a layout page, used by all the other pages.

Required Features:

- The site should have a welcome page, with content free of your own choice. The list page from last weeks assignment should be accessible through a link in a header navigation bar.
- Each person in the list should have a link in the list, linking to a personal details page.
- The header shows a link to a login page. If logged in, a logout link should be presented instead.
- The login link takes you to a page with a form that accepts a simple name/string. The name/string entered should be stored in Context. An entered string counts for being logged in.
- Present the login name/string (if logged in) on the welcome page and in the header.

Code Requirements:

- The person details page should have a specific route adress, made up from data in the person data (Ex. *personlist/firstname_lastname*)
- The site should use a layout page, containing a header (Ex. *"Home", "Personlist",* [login string] *"Login/Logout"*)
- A context provider should be added to the index.js page. A context consumer should be added where needed to fulfill the feature and code requirements.
- Only functional components. For context, use the hook useContext.
- No JavaScript manipulation of the DOM.
- When a React router link is clicked, the web page should not reload but the url should reflect the change

Optional:

- Add buttons for "Previous page" (useHistory)
- Add a theme button that changes the site style (useContext)

Resources:

- https://reactjs.org/docs/hooks-reference.html#usecontext
- https://reactjs.org/docs/context.html
- https://reactrouter.com/en/main

Subjects Covered:

- React
    - React router
    - useContext

React Single-Page CMS

Your assignment is to build a React Single Page Application (SPA) that serves as a Control & Manage System (CMS) for a people list. Your previous assignment will be used to provide the data.

The page should have a list of people, which when clicked on will show detailed information about that person (their name, city, country & languages). This data should be possible to change and doing so should update the list both on the website but also in the database.

Required Features:

- Backend - ASP.NET Core MVC from previous assignment.
  - Create a new controller to be used for your React frontend.
    - Actions shall return JSON and/or http status codes only (no views).
      - Use the services you already have created in the previous assignments.
- Front-End - React
  - One table component that shows all the people in the database.
    - Be able to sort the table content by person's name (with React).
  - One details component that shows all the information related to the person selected from the table.
    - Details shall have a button to delete selected person.
  - One create person component.
    - shall retrieve the list of Countries and Cities and make drop down to select from during creation.

Code Requirements:

- All interactions should use React to change the contents of the page, to prevent refreshing of the page in the browser.
- Data shall be loaded through an AJAX/Axios request and sent to the page as JSON.
- Data should be validated in the frontend before sending it to the backend.
- Data must be in sync between Frontend & Backend.

Optional:

- Add edit for Person.
- Add the same functionalities for City.
- Add the same functionalities for Countries.

Resources:

- https://reactjs.org/docs/thinking-in-react.html
- https://www.taniarascia.com/getting-started-with-react/
- https://app.pluralsight.com/paths/skill/building-web-applications-with-react

React Single-Page CMS

Subjects Covered:

- React
    - Components
    - State & Props
    - Controlling a Form
    - Axios or AJAX
    - Sorting lists

You are going to create a simple JavaScript game, based on the game called ”Sokoban”.

The first task is to create the game board itself. You will be given an array of characters that represents a tile-based map. Your task is to take that character array, and turn it into a grid of html elements that can be used to represent the map, that the player can then move around inside.

The second part of making the game is to create functions to be able to move your player through the game using the arrow keys on the keyboard. The game should respond to inputs without the normal behavior of the arrow keys (i.e. scrolling in the web page).

Required Features:

- Create functions to get the keypresses to move your player, up, down, left and right.
- Create tile-based objects to present different kind of things.
- Create a grid of html elements to build the tile-based map
- A grid of html elements representing a tile-based game board
- A player object that can move between the tiles of the board.
    1. Walls should stop the player
- Movable blocks that can be pushed by the player into empty spaces.
    1. The game should end once all blocks have been pushed into the right spaces on the game board.
- You are NOT allowed to use Canvas.

Code Requirements:

- You must use event listeners to handle key presses to make the player move.
    1. The key press event listener should be able to handle the up, down, left and right arrow keys
    2. The normal effect of those keys should be suppressed, to make sure that the page does not scroll when you press them.
- The grid map must be made up of html elements created through JavaScript commands.
    1. The player and blocks should be represented by html elements.

Resources:

- http://www.w3schools.com/js/ - W3Schools JavaScript Reference
- https://lexicon.udemy.com/the-complete-javascript-course/learn/lecture/5869076#overview – Section 4 is highly relevant
- http://sokoban.info/ - A example of the game in action

Subjects Covered:

- JavaScript interaction with HTML
    1. Creating and modifying elements
    2. Event listeners tied to the HTML document object model.
    3. Making an interactive JavaScript / HTML application

Tips:

- Use CSS to present the different types of tiles on the map (Ex. .player .wall).
- Store the players X & Y position in global variables.
- Use the multidimensional map array index´s as X & Y coordinates and assign them as id´s on the html elements in the map (Ex. id=”x3y7”).
- Build order:
    1. Create html page.
    2. Link in the SokobanBase.js file.
    3. Link in your CSS and JS files.
    4. Create a element with a id to contain your map.
    5. Create a function to generate the elements that will represent the tiles from the map array.
        - Give them a id and CSS.
        - Append them into your map element.
    6. Use an event to listen for key presses on the key board and make it run a function arrowKeys.
    7. Function arrowKeys will check if it was a arrow key that was pressed and if so run a movePlayer function.
        - Send in the change in X & Y coordinates (Ex. ← = X -1, Y 0).
        - Use the player coordinates and the change coordinates to find the element the player wants to move to.
        - Update CSS to move the player around the map.
    8. Start adding one rules at the time for if the player can move or not.

MVC Basics, Guessing Game

Add a new view containing a number-guessing game.

The page should include a form that allows you to input a number. When the page is loaded for the first time, the controller should generate a random number between 1 and 100, that it will save in a session so it remembers it for the page even if it is refreshed.

When you submit a number into the form, it should compare your number to the one the controller generated. If it is the correct one, you should get a message congratulating you on your success, and a new number should be generated.

Otherwise, it should tell you whether your guess is either too high or too low, and let you take another guess.

Required Features:

- A randomly generated number, hidden from the user of the page.
- The ability to guess, through a form, which number was generated.
- A message displaying how well you did.

Optional:

- Add a counter for how many times the user has guessed, and display it after every failed guess. This should reset when the correct number is guessed.
- Add a high score list that is stored in a cookie, and displayed on the page.

Code Requirements:

- The guessing game View should be accessible through a custom route, using the "/GuessingGame" pattern, regardless of what the controller is named.
- The random number should be stored in the Session state.
- Guessing should be handled through overloaded Actions, referring to the same View.
    - When the page is loaded through a GET request (such as through the URL), the app should generate a new number (if none exist) or continue with the existing one.
    - When the page is loaded through a POST request, it should make a guess, unless the value isn't provided or invalid, in which case it should display an error message.

MVC Basics, Guessing Game

Resources:

- https://app.pluralsight.com/courses/72ba6cdd-6f01-4bf1-a17a-37419596f317/table-of-contents

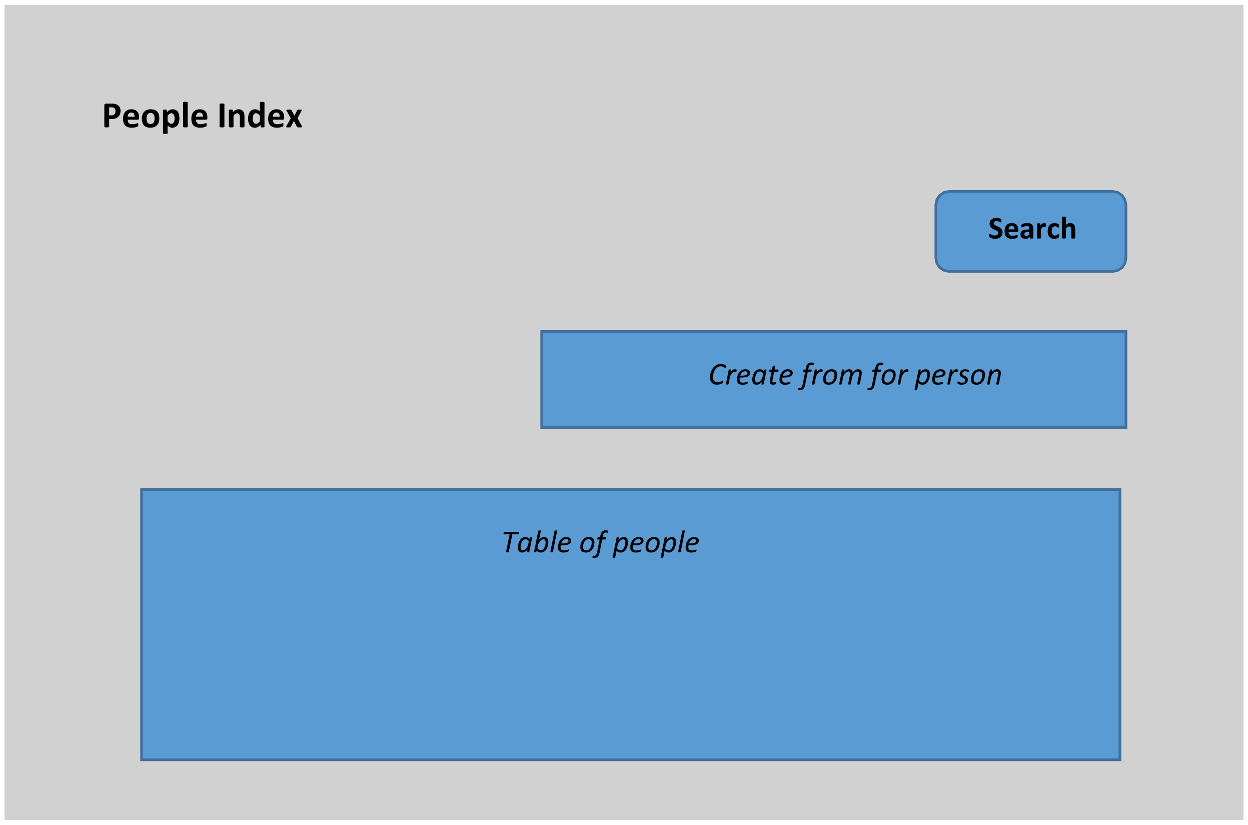
Subjects Covered:

- State management.
    - Session
    - Cookies
- Overloaded actions with GET and POST

Create a MVC project with a controller with one view to displays a list of people.
These people should have a name, phone number and city.
The controller will let a model take care of the business logic and use a ViewModel to send out the needed data to the view.

Required Features:

- A single view that has the following:
    - Html table of people.
        - Each row should show a person, and a link that when clicked, removes that person.
    - Two forms:
        - A form that filters the table content – if you submit the form, the page should be refreshed and only show the people whose names or cities with name containing the string you entered in the form.
        - The other form should let you add a person to the list of people.



Code Requirements:

- Models
    - Person – Person data.
    - CreatePersonViewModel – Use to prevent overposting and to use data annotations to validate inputs when creating new person.
    - PeopleViewModel – container for the information you need in your people view.
- The table data should come from a view model, which should have a list of people, and the search phrase if one exists.

Optional:

- Add buttons to sort the list on the page.
    - Sort in alphabetical order and reverse alphabetical order, by name or by city.
- Add a checkbox which determines whether the filtering should be case sensitive or not.

Resources:

- https://dotnettutorials.net/lesson/view-model-asp-net-core-mvc/

Subjects Covered:

- Models
    - View Models
        - @Model vs @model
        - @using
        - Data annotations
- Forms
    - GET vs POST

MVC Data, Partial Views

In this assignment, you will continue developing the list of people that you added to in the last assignment, changing the general layout and structure of how the people are displayed.

This assignment is preparation for the next assignment where we are going to use those PartialView´s.

Required Features:

- Rework the display of the person list inside the original people Index view to be based on PartialView instead and those be written so they can be used in other scenarios other than being inside a list.

Code Requirements:

- Instead of being rows in a table, each item in the list should be defined as a div element using Bootstrap row css – basically a page fragment on its own that is not dependent being inside a table element anymore.
  - This fragment should be defined inside its own partial view, instead of in the view itself.
  - The partial view should take one instance of the person as its model.
- Use Razor TagHelper < partial />

Resources:

- https://docs.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-6.0
- https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/partial-tag-helper?view=aspnetcore-6.0

Subjects Covered:

- Partial Views with Models

MVC Data, AJAX

In this assignment, you will create a AjaxController that will provide only one view that uses JavaScript jQuery AJAX to make the resulting web page into a SPA(Single Page Application).

Required Features:

- After you have loaded the html page, there shall be no more reloads of the page, changing the page content using JavaScript is okay.
- Three buttons that are hocked up to JavaScript
    1. People – use Ajax Get to fetch the PartialView with list of people and display it on the page using JavaScript.
    2. Details – use an input element so the user can type in an Id of a person and when thy press the button, make an Ajax Post with Id to fetch a PartialView on the person that has the Id and display it on the page using JavaScript.
    3. Delete – use the same input element as Details but when thy press the Delete button, make an Ajax Post with Id to a Delete action that removes the person that has the Id and display a message on the page if person was removed or not on the page using JavaScript.
- The result from pressing any of the three buttons shall be displayed in one element overwriting what was there before.

Code Requirements:

- Create Actions that return PartialView for People button & Details button and return status codes for the Delete action.
- Use Ajax to get/post and sync the frontend/backend by using the results from the controller actions.

Resources:

- https://app.pluralsight.com/library/courses/interacting-data-using-jquery-ajax/table-of-contents
- https://www.c-sharpcorner.com/blogs/model-popup-in-asp-net-core-31

Subjects Covered:

- Partial Views from Actions
- AJAX

Entity Framework Introduction

We will continue from the previous assignment and now change from saving our people in memory to using a database. Entity Framework will be our connection to the database.

Required Features:

- Use Dependency Injection to inject database context into your controller.
- Use the database context to persist our Person data on thedatabase and retrieve the data when needed.
- Use LINQ at least once in a method.
- Use seeding to some extent.
- Data should be persistent between executions of the program.

Code Requirements:

- A database must be created through Entity Framework and connected to corresponding models in the C# code.
- Sending data to and from the database should be done through an Entity Framework database context.
- The context should only contain these sets of data:
  - People

Resources:

- http://www.entityframeworktutorial.net/
  - Entity Framework tutorial site
- https://msdn.microsoft.com/en-us/library/bb397933.aspx
  - LINQ Reference

Prerequisites:Basic understanding of relational databases.

- Basic understanding of SQL queries.

Subjects Covered:

- Entity Framework, Code First

Entity Framework Many-to-One

We will continue from the previous assignment and now add a new Country´s and City´s class´s to be stored alongside the people in the database.

Required Features:

- Controller and ViewModels for Country class.
- Controller and ViewModels for City class.
- Country will have a list of Cities.
- City will have a list of People.
- Replace string City in Person with class City.
- There should be some default (seeded) data in the database tables.
- Data should be persistent between executions of the program.

Code Requirements:

- A database tables must be created through Entity Framework and connected to corresponding models in the C# code.
- Sending data to and from the database should be done through an Entity Framework database context.
- One-to-Many relationships need to be mutual relationships – an object reference on one side, and a list object on the other side.
- The context should only contain these sets of data:
  - People, Country & City

Resources:

- http://www.entityframeworktutorial.net/
  - Entity Framework tutorial site
- https://msdn.microsoft.com/en-us/library/bb397933.aspx
  - LINQ Reference

Prerequisites:

- Previous Entity Framework assignment 4 done.

Subjects Covered:

- Entity Framework
- Code First
- Many-to-One
- One-to-Many

Entity Framework Many-to-Many

We will continue from the previous assignment and now add Language class to be stored alongside the people in the database.

Required Features:

- Controller and ViewModels for language class.
- Language will have a list of people.
- Person will have a list of Languages.
- Should be able to add Languages to a Person.
- There should be some default (seeded) data in the database tables.
- Data should be persistent between executions of the program.

Code Requirements:

- A database tables must be created through Entity Framework, and connected to corresponding models in the C# code.
- Sending data to and from the database should be done through an Entity Framework database context.
- Many-to-Many relationships need to be mutual relationships –a list object on both sides.
- The context should only contain these sets of data:
  - People, Country, City, Language & PersonLanguage

Resources:

- http://www.entityframeworktutorial.net/
  - Entity Framework tutorial site
- https://msdn.microsoft.com/en-us/library/bb397933.aspx
  - LINQ Reference

Prerequisites:

- Previous Many-to-One assignment 5 done.

Subjects Covered:

- Entity Framework
- Code First
- Many-to-Many
- Association table / Joining table

MVC Identity

In this assignment, we will expand the earlier people list to include a Content Management System (a CMS), that will let an authorized user log into the page and edit the data in the database.

Required Features:

- Any visitor to the site should be able to register a user account, and once an account has been created, they should be able to:
    - View the list of people.
    - Add a new person to the people list.
    - Edit existing people.
- User must fill in the following additional information about themselves during registration to be saved in their user information:
    - First name & Last name.
    - Birth date.
- Once an account has been created, it should be possible to flag it as an admin.
    - Admins should be able to do everything normal users can.
    - Admins should be able to view, create, edit and delete people, cities, countries.
- If you are not an admin, the City/Country controller should not be accessible, but they should be available as selectable options in the people pages (Ex: Add person to city).

Code Requirements:

- Use ASP.NET Core Identity to manage roles and accounts.
    - Must create/use your own Models/Views/Controllers.
- Authorization should be done through roles, which are assigned to users.
    - Minimum of two roles are to be used.

Resources:

- https://app.pluralsight.com/library/courses/authentication-authorization-aspnet-core
- https://docs.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-6.0
- https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio

Prerequisites:

- MVC Database, Many to Many

Subjects Covered:

- Authentication through ASP.NET Identity
- Content Management Systems