**DIA SUPERMARKET PROJECT FLOW**

1. Project Idea? eCommerce.
2. Project Category? Fresh Grocery Products.
3. Research on products:
   1. Images on unsplash and pixabay.
   2. Chase.pk for product categories and product details.
   3. Zanjabeel book store for cart, checkout and order confirmation.
4. Tables names, their attributes and relationships.
5. Template choose for front end and admin and download via HTTrack.
6. SQL queries for all tables, apply constraints.
7. Insert dummy records in Admins, Users, Categories and Products tables.
8. Create ASP.NET MVC project:
   1. In Visual Studio, search for “Web Application ASP.Net MVC C#”.
   2. Create new project choosing “Empty” and check on MVC checkbox at right.
9. Model -> Add New Item -> ADO.NET -> New connection > fill in all details, click on yes in case it asks to include sensitive data.
10. Build solution.
11. Set up template:
    1. Create new folder in project named, Templates; add two sub folders in it named: Admin and Front. Copy all resource files like CSS, JS, Images, etc. from websites we downloaded and paste into this folder.
    2. Create a folder “Shared” inside Views folder. Add two sub-folders: Admin and Front. Create \_LayoutBasic files in each folder for navbar/sidebar and footer etc.
12. Create Controllers named as Admin and Home. Create Index () action method in each controller. Add home pages main body in each Index views using layout page (for header footer).
13. Admin Registration:
    1. Add two Register methods with HttpGet and HttpPost (Post one will be having submitted form as param).
    2. Set Register view from templates (we used 3 pages from admin template: register, form-basics, form-pickers).
    3. Convert form controls into Html Helper Methods of ASP.NET like form -> @using (Html.BeginForm) and <input> into @Html.EditorFor() etc.
    4. Write submission code in controller in Register HttpPost inside ModelState.IsValid.
    5. Comment out HTML validations and apply Data Annotations in tb\_Admins models:
       1. Add Required, StringLength, Range etc. annotations along with custom error messages.
       2. In Register view, add @Html.ValiationMessageFor line for displaying error messages
       3. Confirm Password annotation in Model and add new control for it in Register Form
       4. Email already exists validation (code it in controller).
       5. Return and show success message on registration successful.
14. Admin Login:
    1. Right click on Models folders and add new class named “LoginModel” and copy two fields email and password (from tb\_Admins class) along with Data Annotations.
    2. In AdminController write two methods for Login, one HttpGet one and other HttpPost one with LoginModel object as parameter.
    3. Set view file according to needs.
    4. Convert html to html helper methods of form and also add ValidationMessageFor field for both controls. (don’t miss out @model ProjectName.LoginModel at top)
    5. **Above** **@using (Html.BeginForm)** add the code given below. This will later on print message “Invalid email or password.”; as this error is not for specific control that’s why we add this above form.

<p class="text-danger small font-weight-bold text-center">@Html.ValidationMessage("InvalidLoginError")</p>

* 1. Write login logic in HttpPost action method of login in controller.
  2. Add [ValidateAntiForgeryToken] after [HttpPost]; also, inside curly brackets of @using (Html.BeginForm) { .. } add this: @Html.AntiForgeryToken()
  3. In web.config file, ctrl+F for system.web tag and add this inside for authentication:

<authentication mode="Forms"><forms loginUrl="/Admin/Login"></forms></authentication>

* 1. Now run application, fill form with some email and password from Database and check if it’s taking you back to index page or not.
  2. Do the same for front user.

1. Welcome, Wajiha in navbar (layout page) and index page.
2. If user is logged-in, login and register page should not be seen until logout. Similarly, if the user is not logged in, he should not see any dashboard etc.
   1. In controller set this via Redirect to action by checking if session variable is null or not.
3. Logout: kill session variable and authentication forms. And set logout link in navbar.
4. Set template things!
   1. Logo designed: <https://www.canva.com/>
   2. Image background remover: <https://www.remove.bg/>
   3. Un-minification of CSS, JS: <https://unminify.com/>
   4. Color picker: Firefox’s default
5. Admin Edit Profile:
   1. Action method HttpGet:
      1. if user is not logged-in, redirect to Login page.
      2. Find user in database by session id and show send object on view to display details.
   2. View:
      1. Copy content from register page. Change headings etc. and submission method in begin form.
      2. Add hidden field for id and values you don’t wanna change.
      3. Make email readonly field.
   3. Action method HttpPost:
      1. Update via db.Entry. Also, change session variable in case user tries to change name.
      2. Send success message and add notification code in view.
6. Shop Page:
   1. Install – PagedList and PagedList.Mvc in project. (Right click on project name in solution explorer / Manage Nuget Packages / Browse Tab / Search, install and build project)
   2. Add model class: Products\_with\_Filter with two properties; one for for paginated products list and other for applying filters (add using PagedList at top).

public PagedList.IPagedList<tb\_Products> ProductsList { get; set; }

public tb\_Products ProductForFilter { get; set; }

* 1. In HomeController, add using *PagedList;* at top and then write Shop action method.
     1. Create products list using PagedList in object of class we just made in controller.
  2. Add view (Model we created and Layout front).
     1. @using PagedList.Mvc at top.
     2. Copy html content from shop page of template. Change images references.
     3. Out of 12, keep only 1 product div – put that one div inside foreach loop of collection productslist we sent from controller.
     4. There is no place to show category of product in html so add it yourself.
     5. Insert products into DB using insert into queries.
     6. Convert html with Html.DisplayFor for all values of product from table.
  3. View - Product’s Pagination:
     1. Write pagination code at bottom
     2. For distance between page number circle change CSS.

1. Admin – Add Products:
   1. Action Method – HttpGet:
      1. Work for dropdown list of categories names.
      2. Redirection in case not logged in.
   2. Data annotations in products table model.
   3. View - any form from template. Convert html into Html Helper methods. (I had to add jQuery code in layout page for upload image box).
   4. Write HttpPost with select box having selected category, code for unique product name, extra fields manually added, save product in DB and success message.
2. Products List:
   1. Action method: add using System.Data.Entity; for including category table with Products because of FK relation.
   2. View:
      1. Template set
      2. Model as IEnumberable
      3. Keep one tr in foreach loop and convert html to html helper methods.
      4. Set view, edit, delete links.
      5. Change Data Table settings in Templates/Admin/vendors/scripts/dashboard.js -> ctrl+F “data-table” and set properties to true as per need.
3. Delete Product:
   1. Action method - HttpGet: handle null and fetch product to display details.
   2. View – wrote table, changed css a bit too in style.css file.
   3. View – confirm delete via modal box.
   4. Action method – HttpPost with name ConfirmDeleteProduct.
4. Product Details / View Product:
   1. Action method – HttpGet only and exactly as delete’s HttpGet one.
   2. View – exactly as Delete’s with some button and html changes.
5. Edit Product:
   1. Action method – HttpGet: exactly as delete HttpGet’s one except select list for categories.
   2. View - exactly as Add Product except add hidden field for id and some html changes.
   3. Action method – HttpPost:
      1. check for unique name other than entered id (in case user doesn’t want to change product name)
      2. Update each field one by one and save changes to db.
6. Image upload and Show
   1. Html.BeginForm("AddProduct", "Admin", FormMethod.Post, new { enctype = "multipart/form-data" }
   2. Input type=”file” name=” customFile” id=”customFile”
   3. Action method HttpPost, add second param as HttpPostedFileBase customFile
   4. Create a folder “UploadedFiles” in Templates folder.
   5. Write UploadImage method.
   6. Call UploadImage method in add product HttpPost action method based on if else.
      1. In Add, send file only (store file if it doesn’t exist already else give error)
      2. In Edit, send file and DB saved file name to check if it already exists what to do (2 cases: 1. Delete image and upload fresh one. 2. Don’t delete or upload image but only return success)
      3. To use this, make 2 methods 1. isFileALreadyExist() 2. DeleteImage()
   7. Write code to display image right away when user upload image in View.
   8. Fill image from DB in Views like shop, products list, edit, add, view, delete pages.
7. Do Front user login and register work. (leave form authentication part in login)
   1. Model changes + controller methods + view.
8. Added 22/b/v
9. Template changed of front with all static page’s addition.
10. Add image field in category table in database + add verified\_user column in Admin table.
    1. Backup all model files we changed or added data annotations.
    2. Open .edmx diagram in VS
    3. Right click / Update Model / Refresh tab / select tables we changed from dropdown
    4. Finish + build project
    5. Add Data annotations again from backup.
    6. Run project and test if everything works fine.
11. Only verified users can login in admin:
    1. Add verified user condition in login method and change error message accordingly.
12. Home Page Front
    1. Fill products with DB.
13. Single product page.
    1. Action method + view.
    2. Details with model binding.
    3. Related products with ViewBag.
    4. Link shop, home page, single products’ related product clickable link to action link of single page.
    5. Complete all queries to be shown on front with SQL and LINQ.
14. Shop page categories fill by DB.
    1. Filtering of categories is working fine but only without page number.
15. Admin dashboard filled by DB queries via ViewBag.
    1. Default order of sorting set in admin dashboard.js data-table class. (had to add class in products list page too)
16. Navbar for logged-in vs logged-out users in front.
    1. Make it under if else checking session variable.
17. Authentication
    1. Admin – add [Authorize] to the one you don’t want to give access.
    2. User
       1. Create new folder Common and create new attribute in that.
       2. User controller – add [AuthorizeUser] to the one you don’t want to give access.