Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г. В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

Специальность: 09.02.07 Информационные системы и программирование

Квалификация: Программист

ПРАКТИЧЕСКИЕ РАБОТЫ ПО ДИСЦИПЛИНЕ

«РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ»

Листов: 8

|  |  |
| --- | --- |
| Выполнил студент | Проверил преподаватель |
| Группы П-2-24  Афанасьев.А.С | Осипян Л.Г |
|  |  |

Москва 2025

Цель работы: Разработать веб-приложение на платформе ASP.NET Core MVC без использования базы данных, демонстрирующее работу серверной части, обработку HTTP-запросов и взаимодействие клиента с сервером. Приложение должно включать навигацию между страницами, главную страницу, страницу профиля с формой ввода данных и их отображением, а также дополнительную страницу. Необходимо реализовать кастомизацию стилей и навигационного меню в соответствии с архитектурой MVC.

На скриншоте показан файл-контроллер "HomeController", который обрабатывает все запросы пользователя. В нём есть пять методов: "Index()" для главной страницы, "Profile()" для отображения профиля, "SaveProfile()" для сохранения данных из формы, "ClearProfile()" для очистки профиля и "About()" для страницы "О приложении". Данные пользователя хранятся в статической переменной "\_userData" типа "Dictionary<string, string>" — это словарь, где ключи это названия полей ("name", "age" и т.д.), а значения — введённые пользователем данные. Такой подход позволяет обойтись без базы данных, как того требует задание. При сохранении данных сначала очищается старый словарь методом "Clear()", затем добавляются новые значения с проверкой на пустоту через "string.IsNullOrWhiteSpace()", чтобы не сохранять пустые поля. После сохранения происходит перенаправление на страницу профиля через "RedirectToAction("Profile")", что предотвращает повторную отправку формы при обновлении страницы. Методы "SaveProfile()" и "ClearProfile()" помечены атрибутом "[HttpPost]", что означает их работу только с POST-запросами (отправка форм), в то время как остальные методы обрабатывают обычные GET-запросы (открытие страниц).

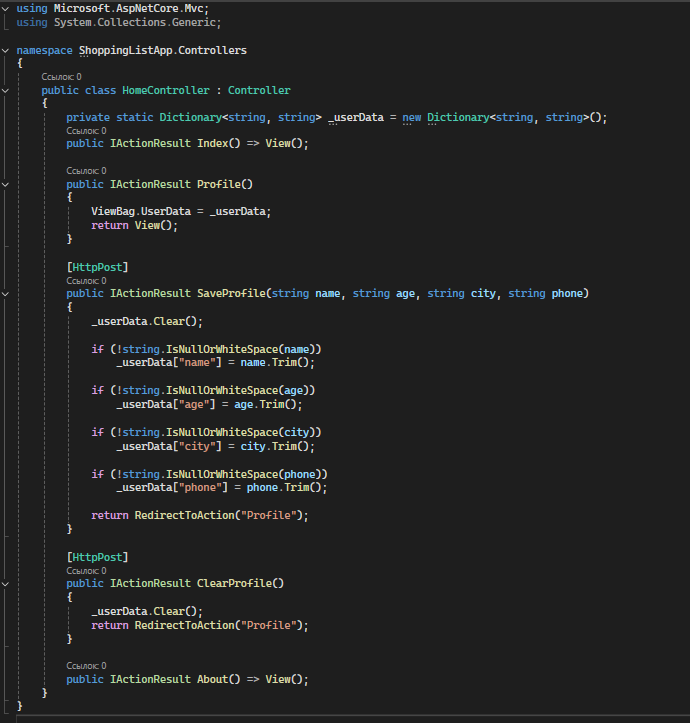


Рисунок 1 – контролер

На скриншоте показано представление страницы "О приложении" — файл "About.cshtml". В самом верху находится блок кода на языке: "@{ ViewData["Title"] = "О приложении"; }", который устанавливает заголовок страницы для отображения во вкладке браузера. Далее идёт разметка с заголовком "О приложении" внутри блока с классом "page-header", который центрирует текст и добавляет стилизацию. Основной контент страницы находится внутри контейнера "about-container", который ограничивает ширину и выравнивает элементы по центру. На странице размещены карточки с информацией: первая карточка "about-card" содержит описание приложения и иконку в виде "✨", вторая секция "features-cards" показывает два блока с возможностями приложения (создание профиля и безопасность), каждый со своей иконкой ("👤" и "🔒") и списком преимуществ через теги "ul" и "li". В самом низу размещена призывная карточка с градиентным фоном и кнопкой "Перейти к профилю", которая использует атрибут "asp-controller="Home" asp-action="Profile"" для перехода на страницу профиля.

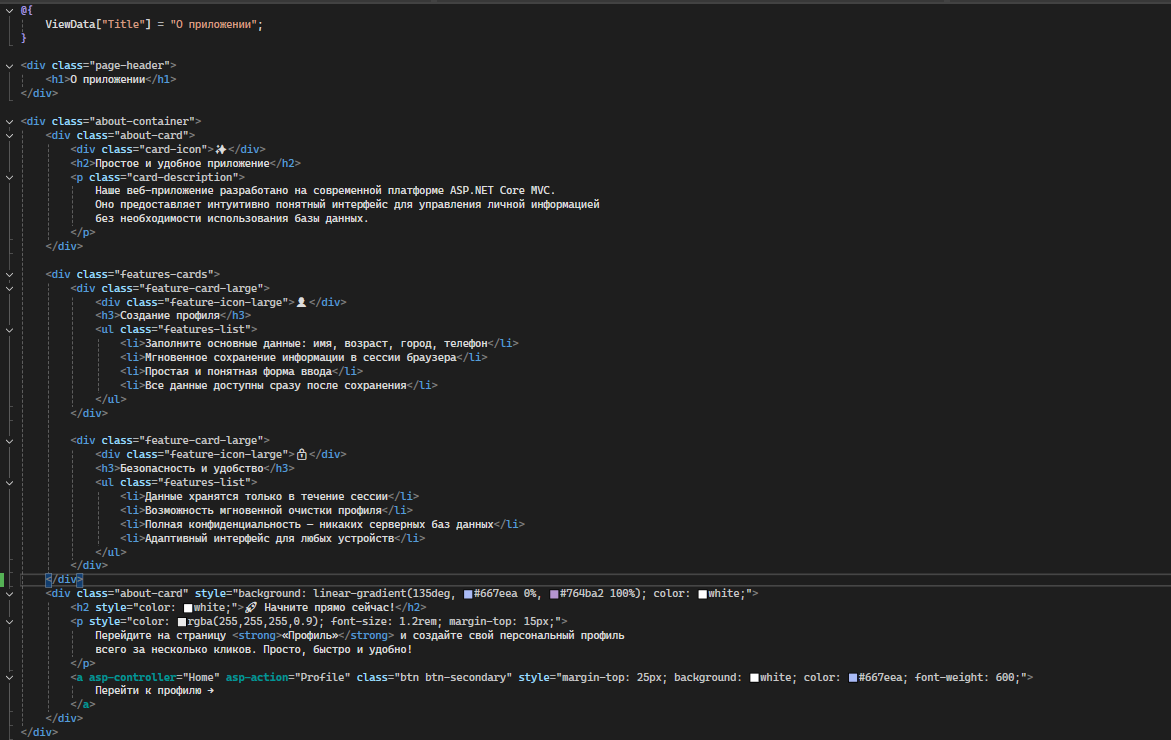


Рисунок 2 – О приложении

На скриншоте показан фрагмент кода представления, реализующий навигационные кнопки в шапке приложения. Первый элемент "<a asp-controller="Home" asp-action="Profile" class="btn btn-primary">Начать</a>" создает кнопку перехода на страницу профиля, где "asp-controller" указывает контроллер "Home", а "asp-action" — метод "Profile". Второй элемент "<a asp-controller="Home" asp-action="About" class="btn btn-secondary">Узнать больше</a>" формирует ссылку на страницу "О приложении". Классы "btn", "btn-primary" и "btn-secondary" определяют стили кнопок через CSS, обеспечивая визуальное разделение основного и второстепенного действия.

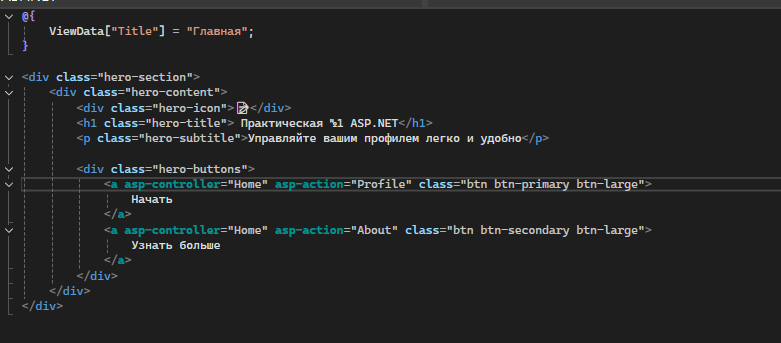


Рисунок 3 - Главная

На скриншоте показан код представления страницы "Профиль", реализованный на языке Razor. В начале файла устанавливается заголовок страницы через "ViewData["Title"] = "Профиль";", а затем объявляется переменная "userData" для получения данных из "ViewBag.UserData". Страница разделена на две колонки с помощью контейнера "profile-container" и сетки "profile-grid". Левая колонка содержит форму с четырьмя полями ввода: "Имя" (тип "text" с валидацией только на буквы через атрибут "pattern"), "Возраст" (тип "number" с ограничениями "min=1" и "max=120"), "Город" (аналогично имени) и "Телефон" (тип "tel" с маской ввода). Обязательные поля помечены символом "\*" через класс "required". Форма отправляется методом "POST" на действие "SaveProfile" контроллера "Home". Правая колонка отображает сохраненные данные: если словарь "userData" содержит непустые значения, они выводятся в виде списка с заголовками (например, "<strong>Имя:</strong> значение"), иначе показывается блок "empty-state" с сообщением "Профиль пуст". Внизу правой секции расположена кнопка "Очистить профиль", отправляющая "POST"-запрос на метод "ClearProfile".

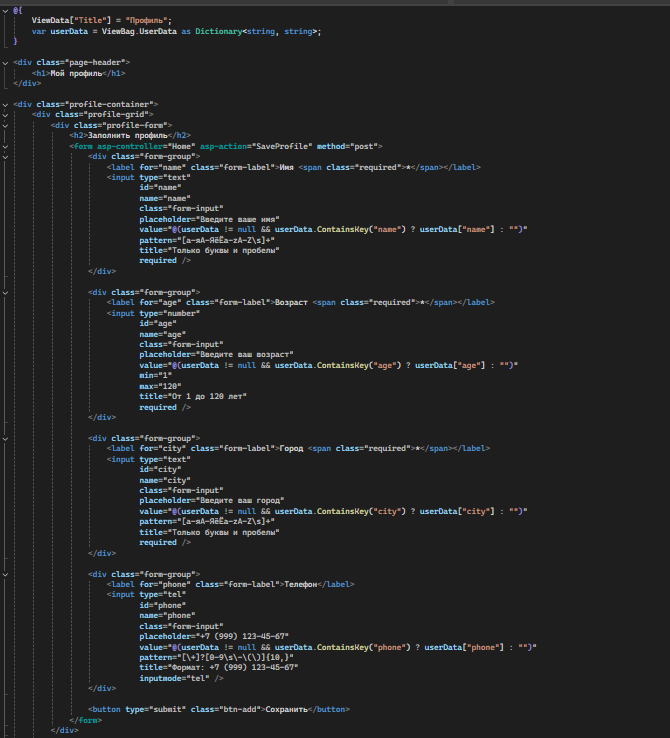


Рисунок 4 - Код профиля ч1

На скриншоте показан второй фрагмент кода представления профиля, который проверяет наличие данных в словаре "userData". Если данные присутствуют ("userData != null && userData.Count > 0"), отображается список с информацией: для каждого поля ("name", "age", "city", "phone") проверяется, что ключ существует в словаре и значение не пустое через конструкцию "@if (userData.ContainsKey("name") && !string.IsNullOrWhiteSpace(userData["name"]))". При соблюдении условия выводится строка вида "<strong>Имя:</strong> @userData["name"]". Если данные отсутствуют, показывается сообщение "Профиль пуст" с иконкой и подсказкой "Заполните данные!". Внизу размещена форма с кнопкой "Очистить профиль", отправляющая POST-запрос на метод "ClearProfile" контроллера "Home"

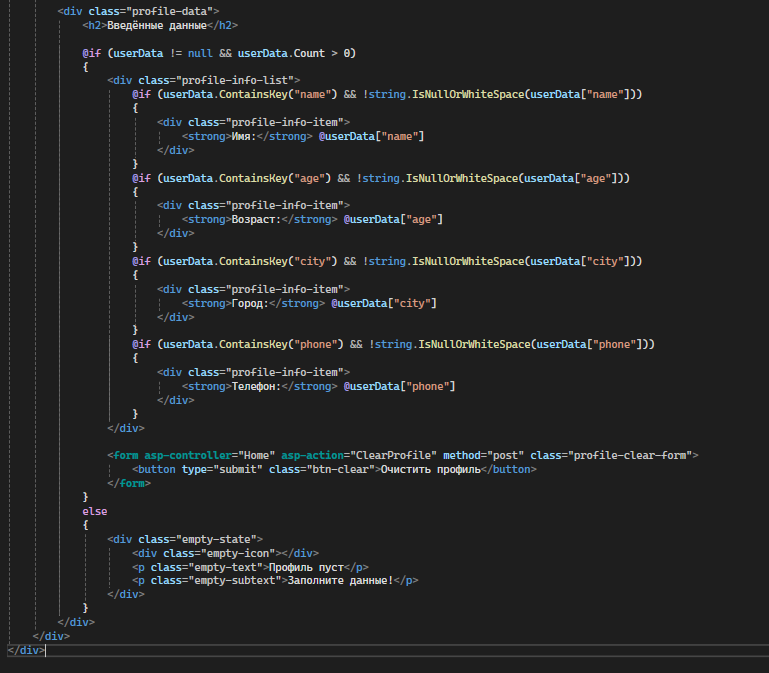


Рисунок 5 - Код профиля ч2

В файле “\_Layout” были изменены пути в навигации и надпись в футере.

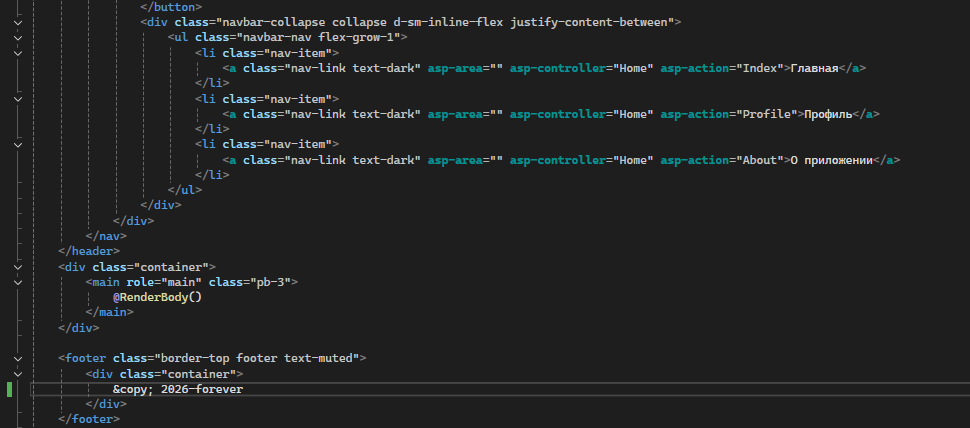


Рисунок 6 - Лайаут

В файле “Program.cs” никаких изменений не внесено т.к. дефолтные файлы не были переименованы.

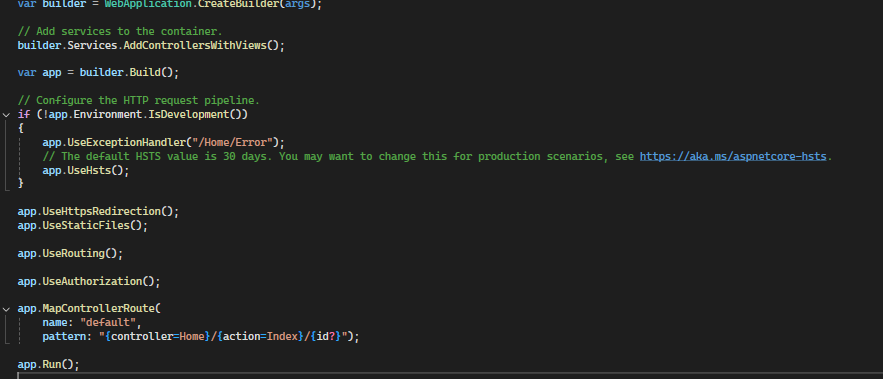


Рисунок 7 - Программ

Та же полено знать следующую информацию которая поможет лучше понять работу проекта:

**Разница между HTTP и HTTPS:**

HTTP (Hypertext Transfer Protocol) — это протокол для передачи данных между браузером и сервером. Он работает в открытом виде, то есть все данные (включая пароли и личную информацию) передаются в виде обычного текста, который можно перехватить. HTTPS (Hypertext Transfer Protocol Secure) — это защищенная версия HTTP, которая использует шифрование через SSL/TLS. Это значит, что все данные между браузером и сервером шифруются, и даже если кто-то перехватит трафик, прочитать его будет невозможно. В современных приложениях почти всегда используется HTTPS для безопасности.

**Атрибуты [HttpGet] и [HttpPost]:**

В коде контроллера методы помечаются атрибутами [HttpGet] и [HttpPost], чтобы указать, какой тип HTTP-запроса они обрабатывают. [HttpGet] обрабатывает запросы на получение данных (когда пользователь просто открывает страницу в браузере), а [HttpPost] — запросы на отправку данных (когда пользователь заполняет форму и нажимает кнопку "Отправить").

В коде на скриншоте методы "SaveProfile" и "ClearProfile" помечены как [HttpPost], потому что они обрабатывают отправку формы — первый сохраняет данные профиля, второй очищает их. Метод "Profile" не имеет атрибута, потому что по умолчанию все методы контроллера считаются [HttpGet], если не указано иное. Это значит, что при открытии страницы "/Home/Profile" в браузере вызовется именно этот метод для отображения страницы.

Вывод: В ходе работы было разработано веб-приложение "Профиль пользователя" на ASP.NET Core MVC. Приложение включает три страницы: главную, профиль и "О приложении". На странице профиля реализована форма с полями для ввода имени, возраста, города и телефона, с валидацией данных на стороне клиента и сервера. Данные сохраняются в статической переменной контроллера, что позволило обойтись без базы данных. Навигация между страницами реализована через меню в шапке приложения. Интерфейс полностью кастомизирован с использованием собственных CSS-стилей. Приложение корректно обрабатывает GET и POST запросы, использует паттерн перенаправления после отправки формы и стабильно работает. Все требования задания выполнены.