Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра Информатики
Специальность «ИиТП»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
«Работа с базой данных MySql»

БГУИР КП 1–40 04 01 ПЗ

Выполнил:
студент группы 753503
Муха Д.С.

Руководитель:
Удовин И. А.

Минск 2020

Оглавление

Введение

В данном курсовом проекте работа с базой данных MySql будет проводиться на базе сайта подбора временного жилья под названием “House Rent” наподобие “Booking” или “AirBnB”.

1. Инструменты разработки

1.1. Назначение разработки

House rent - это проект позволяющий людям бронировать и снимать временное жилье на территории Беларуси.

1.2. Перечень основных выполняемых функций

House rent разделяет пользователей на две группы:

- Обычный пользователей (normal) – может авторизоваться, просматривать существующее доступное жилище, а также выставлять на съем собственное.
- Админ (admin) – может авторизоваться, просматривать существующее доступное жилище, выставлять на съем собственное, а также управлять пользователями и публикациями с жильем.

При создании поста с жильем, отправляется запрос на размещение администратору. Он решает, давать разрешение на публикацию или нет. Как только администратор выставляет решение, пользователю, сделавшему запрос, высылается письмо на почту с уведомлением о статусе его запроса.

При бронировании жилья арендодателю высылается письмо на почту с уведомлением о заявки на бронь. Как только заявка одобрена или отклонена, высылается письмо на почту пользователя, сделавшего запрос на бронь, с ответом.

1.3. Входные и выходные данные пользователя

Пользователь должен иметь возможность зайти в систему используя уникальный никнейм и пароль, который должен соответствовать определенным требованиям (длина более 8 символов, символы английского алфавита и цифры из которых обязательно должно быть: хотя бы 1 буква в верхнем регистре, 1 в нижнем, 1 цифра, 1 не буквенно-циферный символ)

1.4. Требования к надежности

Обычный пользователь (user) не должен иметь возможность напрямую влиять на содержание поста, а также не должен иметь возможность изменять свою роль, роли других пользователей, а также создавать новые посты.

1.5. Требования к эргономике

Пользовательский интерфейс сайта должен отвечать современным требованиям к эргономике и технической эстетике.
Внутренний пользовательский интерфейс сайта должен позволять администратору относительно свободно ориентироваться в информационном и функциональном пространстве системы и удовлетворять следующим требованиям:

1. Однозначно понимаемое назначение названий пунктов меню (функциональных элементов интерфейса) или их графических изображений.

2. Группировка элементов интерфейса по функциональному признаку.

3. Минимизация вертикальной и горизонтальной прокрутки.

4. Четко сформулированные и понятные пользователю сообщения об ошибках.

5. Цветовая гамма, выдержанная в спокойных тонах (или, как минимум, не должна идти кровь из глаз при просмотре интерфейса), не раздражающих пользователя.

Внешний пользовательский интерфейс сайта должен удовлетворять следующим требованиям:

1. Адекватно отображаться в зависимости от типа подключения

пользователя (пользователи, работающие через модем;

пользователи, работающие через высокоскоростной канал

доступа).

2. Обеспечивать большую скорость загрузки страниц, достигаемую

в результате оптимизации графических элементов.

3. Обеспечивать минимум усилий и временных затрат пользователя для навигации по страницам сайта.

4. Корректно отображаться при различных разрешениях и количестве одновременно отображаемых цветов монитора.

5. Сохранять идентичность отображения на большинстве современных ОС и Web-браузерах (Mozilla Firefox, начиная с первой версии Quantum и до текущей версии; Opera, начиная с версии 9 и до текущей версии; Google Chrome начиная с версии 70 и до текущей версии)

1.6. Требования к организации системы

Система должна состоять из мало зависимых подсистем (отдельных приложений), которые обеспечивают работу всей системы в целом. Данные подсистемы должны реализовывать свой функционал и развиться как самостоятельные подсистемы.

Можно использовать классический паттерн проектирование MVC(Model-View-Controller).

Статическая страница на HTML не умеет реагировать на действия пользователя. Для двустороннего взаимодействия нужны динамические веб-страницы. MVC — ключ к пониманию разработки динамических веб-приложений, поэтому разработчику нужно знать эту модель.

MVC расшифровывается как модель-представление-контроллер. Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения. Компоненты MVC:

- Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

- Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.
- Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

1.7. Обоснование выбора языка программирования и среды разработки

В качестве языка программирования был выбран C#, в качестве среды разработки – Visual Studio Community 2019. Эта среда разработки, как и сам язык создана компанией Microsoft, следовательно, писать код получается максимально нативно.

2. Моделирование предметной области

2.1. Обзор используемых технологий

2.1.1. C#

C# (произносится си шарп) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Из-за технических ограничений на отображение (стандартные шрифты, браузеры и т. д.) и того, что знак диеза ♯ не представлен на стандартной клавиатуре компьютера, при записи имени языка программирования используют знак решётки (#). Это соглашение отражено в Спецификации языка C# ECMA-334. Тем не менее, на практике (например, при размещении рекламы и коробочном дизайне), «Майкрософт» использует знак диеза.

Названия языков программирования не принято переводить, поэтому язык называют, используя транскрипцию, — «Си шарп».

C# разрабатывался как язык программирования прикладного уровня для CLR и, как таковой, зависит, прежде всего, от возможностей самой CLR. Это касается, прежде всего, системы типов C#, которая отражает BCL. Присутствие или отсутствие тех или иных выразительных особенностей языка диктуется тем, может ли конкретная языковая особенность быть транслирована в соответствующие конструкции CLR.

Так, с развитием CLR от версии 1.1 к 2.0 значительно обогатился и сам C#; подобного взаимодействия следует ожидать и в дальнейшем (однако, эта закономерность была нарушена с выходом C# 3.0, представляющего собой расширения языка, не опирающиеся на расширения платформы .NET).

CLR предоставляет C#, как и всем другим .NET-ориентированным языкам, многие возможности, которых лишены «классические» языки программирования. Например, сборка мусора не реализована в самом C#, а производится CLR для программ, написанных на C# точно так же, как это делается для программ на VB.NET, J# и др.

C# стандартизирован в ECMA (ECMA-334) и ISO (ISO/IEC 23270).

Известно как минимум о трёх независимых реализациях C#, базирующихся на этой спецификации и находящихся в настоящее время на различных стадиях разработки:

- Mono, начата компанией Ximian, продолжена её покупателем и преемником Novell, а затем Xamarin.
- dotGNU и Portable.NET, разрабатываемые Free Software Foundation.

2.1.2. ASP.Net Core 2.2

Платформа ASP.NET Core представляет технологию от компании Microsoft, предназначенную для создания различного рода веб-приложений: от небольших веб-сайтов до крупных веб-порталов и веб-сервисов.

С одной стороны, ASP.NET Core является продолжением развития платформы ASP.NET. Но с другой стороны, это не просто очередной релиз. Выход ASP.NET Core фактически означает революцию всей платформы, ее качественное изменение.

Разработка над платформой началась еще в 2014 году. Тогда платформа условно называлась ASP.NET vNext. В июне 2016 года вышел первый релиз платформы. А в декабре 2019 года вышла версия ASP.NET Core 3.1, которая собственно и будет охвачена в текущем руководстве.

ASP.NET Core теперь полностью является opensource-фреймворком. Все исходные файлы фреймворка доступны на GitHub.

ASP.NET Core может работать поверх кросс-платформенной среды .NET Core, которая может быть развернута на основных популярных

операционных системах: Windows, Mac OS, Linux. И таким образом, с помощью ASP.NET Core мы можем создавать кросс-платформенные приложения. И хотя Windows в качестве среды для разработки и развертывания приложения до сих пор превалирует, но теперь уже мы не ограничены только этой операционной системой. То есть мы можем запускать веб-приложения не только на ОС Windows, но и на Linux и Mac OS. А для развертывания веб-приложения можно использовать традиционный IIS, либо кросс-платформенный веб-сервер Kestrel.

Благодаря модульности фреймворка все необходимые компоненты веб-приложения могут загружаться как отдельные модули через пакетный менеджер Nuget. Кроме того, в отличие от предыдущих версий платформы нет необходимости использовать библиотеку System.Web.dll.

ASP.NET Core включает в себя фреймворк MVC, который объединяет функциональность MVC, Web API и Web Pages. В предыдущих версии платформы данные технологии реализовались отдельно и поэтому содержали много дублирующей функциональности. Сейчас же они объединены в одну программную модель ASP.NET Core MVC. А Web Forms полностью ушли в прошлое.

Кроме объединения вышеупомянутых технологий в одну модель в MVC был добавлен ряд дополнительных функций.

Одной из таких функций являются тэг-хелперы (tag helper), которые позволяют более органично соединять синтаксис html с кодом C#.

ASP.NET Core характеризуется расширяемостью. Фреймворк построен из набора относительно независимых компонентов. И мы можем либо использовать встроенную реализацию этих компонентов, либо расширить их с помощью механизма наследования, либо вовсе создать и применять свои компоненты со своим функционалом.

Также было упрощено управление зависимостями и конфигурирование проекта. Фреймворк теперь имеет свой легковесный контейнер для внедрения зависимостей, и больше нет необходимости применять сторонние контейнеры, такие как Autofac, Ninject. Хотя при желании их также можно продолжать использовать.

В качестве инструментария разработки мы можем использовать последние выпуски Visual Studio, начиная с версии Visual Studio 2015. Кроме того, мы можем создавать приложения в среде Visual Studio Code,

которая является кросс-платформенной и может работать как на Windows, так и на Mac OS X и Linux.

Если суммировать, то можно выделить следующие ключевые отличия ASP.NET Core от предыдущих версий ASP.NET:

- Новый легковесный и модульный конвейер HTTP-запросов
- Возможность развертывать приложение как на IIS, так и в рамках своего собственного процесса
- Использование платформы .NET Core и ее функциональности
- Распространение пакетов платформы через NuGet
- Интегрированная поддержка для создания и использования пакетов NuGet
- Единый стек веб-разработки, сочетающий Web UI и Web API
- Конфигурация для упрощенного использования в облаке
- Встроенная поддержка для внедрения зависимостей
- Расширяемость
- Кроссплатформенность: возможность разработки и развертывания приложений ASP.NET на Windows, Mac и Linux
- Развитие как open source, открытость к изменениям

Эти и другие особенности, и возможности стали основой для новой модели программирования.

Новое в версии ASP.NET Core 2.2:

- улучшение работы с Web API
- микросервисы и Azure
- улучшение производительности
- поддержка многоэтапной JIT-компиляции
- поддержка расширений SQL Server и SQLite для EF Core
- Обновленные шаблоны Bootstrap и Angular, версий 4 и 6 соответственно
- добавление поддержки API Security + включая HTTP/2 для всех соединений ASP.NET Core.

2.1.3. RazorPages

Razor Pages — новый инструмент, появившийся в Core.Net 2.0. Razor Page — это страница, состоящая из стандартной разметки (View) и бэкенд класса. В каком-то смысле напоминает Web Forms только без поддержки сохранения состояния. Преимущество такого решения очевидно — мы

избавляемся от ненужной прослойки — модели страницы (модель данных в виде например Entity это само собой). Бэкенд страницы является и контроллером и моделью — классика ООП — инкапсуляция данных и методов работы с ними в одном объекте. В конце концов модель страницы — это просто класс, нет никаких причин почему этим классом не может быть контроллер.

Иными словами, Razor Pages — более вменяемое решение для веба чем MVC, теперь мы имеем дело с традиционным и логичным понятием «страница» а не с контролерами и моделями разбросанными по всему проекту. Но поскольку .NET будет развиваться по направлению Core.Net то вряд ли Razor Page появятся в стандартном фреймворке, несмотря на то что ближайшие годы большинство проектов будет оставаться на стандартном .NET. Тем не менее можно изобразить функциональность Razor Pages и на стандартном фреймворке.

2.1.4. Git / GitHub

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удаленный доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и дает возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранятся операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы.

Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создает в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создается рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git свое приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге

.git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит». Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

Репозиторий Git бывает локальный и удаленный. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone.

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создает новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально.

Команда push передает все новые данные (те, которых еще нет в удалённом репозитории) из локального репозитория в репозиторий удаленный. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удаленной копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным двухпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешенные.

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех измененных файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удаленного репозитория на момент завершения и исполнения последней команды pull или push.

Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

- Прямо на сайте можно просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.

- Можно создавать приватные репозитории, которые будут видны только вам и выбранным вами людям. Раньше возможность создавать приватные репозитории была платной.
- Есть возможность прямого добавления новых файлов в свой репозиторий через веб-интерфейс сервиса.
- Код проектов можно не только скопировать через Git, но и скачать в виде обычных архивов с сайта.
- Кроме Git, сервис поддерживает получение и редактирование кода через SVN и Mercurial.

2.1.5. SQLite

SQLite - это встроенная библиотека, которая реализует автономный, безсерверный, нулевой конфигурации, транзакционный механизм СУБД SQL. Это база данных, которая настроена на нуль, что означает, как и другие базы данных, которые вам не нужно настраивать в вашей системе.

SQLite не является автономным процессом, как другие базы данных, вы можете связать его статически или динамически в соответствии с вашим требованием с вашим приложением. SQLite напрямую обращается к своим файлам хранения.

Особенности SQLite

- SQLite не требует отдельного процесса сервера или системы для работы (без сервера).
- SQLite поставляется с нулевой конфигурацией, что означает отсутствие необходимости в настройке или администрировании.
- Полная база данных SQLite хранится в одном кросс-платформенном диске.
- SQLite очень маленький и легкий, менее 400KiB полностью сконфигурированный или менее 250KiB с дополнительными функциями, опущенными.
- SQLite является автономным, что означает отсутствие внешних зависимостей.
- SQLite-транзакции полностью совместимы с ACID, обеспечивая безопасный доступ к нескольким процессам или потокам.
- SQLite поддерживает большинство функций языка запросов, найденных в стандарте SQL92 (SQL2).

- SQLite написан на ANSI-C и предоставляет простой и простой в использовании API.
- SQLite доступен в UNIX (Linux, Mac OS-X, Android, iOS) и Windows (Win32, WinCE, WinRT).

Стандартные команды SQLite для взаимодействия с реляционными базами данных аналогичны SQL. Это CREATE, SELECT, INSERT, UPDATE, DELETE и DROP.

2.2. Информация о базе данных

База данных : HouseRentContext

Таблица 1. Advertise

```
CREATE TABLE [dbo].[Advertise] (
    [ID]                 INT            IDENTITY (1, 1) NOT NULL,
    [Address]            NVARCHAR (MAX) NOT NULL,
    [Category]           NVARCHAR (MAX) NOT NULL,
    [ConfirmationStatus] NVARCHAR (MAX) NULL,
    [FlatDetails]        NVARCHAR (MAX) NOT NULL,
    [FlatSize]           INT            NOT NULL,
    [FlatType]           NVARCHAR (MAX) NOT NULL,
    [Heading]            NVARCHAR (MAX) NULL,
    [OtherBill]          INT            NOT NULL,
    [Phone]              NVARCHAR (MAX) NOT NULL,
    [PostTime]           DATETIME2 (7)  NOT NULL,
    [Rent]               INT            NOT NULL,
    [UserMail]           NVARCHAR (MAX) NULL,
    [UtilitiesBill]      INT            NOT NULL,
    [YoutubeLink]        NVARCHAR (MAX) NULL,
    CONSTRAINT [PK_Advertise] PRIMARY KEY CLUSTERED ([ID] ASC)
);
```

Таблица 2. AdvertiseRequest

```
CREATE TABLE [dbo].[AdvertiseRequest] (
    [ID]     INT            IDENTITY (1, 1) NOT NULL,
    [AdvID]  INT            NOT NULL,
    [From]   INT            NOT NULL,
    [Status] NVARCHAR (MAX) NULL,
    [To]     INT            NOT NULL,
    [Type]   NVARCHAR (MAX) NULL,
    [UserID] INT            NULL,
    CONSTRAINT [PK_AdvertiseRequest] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [FK_AdvertiseRequest_Advertise_AdvID] FOREIGN KEY
([AdvID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE,
```

```
    CONSTRAINT [FK_AdvertiseRequest_User_UserID] FOREIGN KEY ([UserID])
REFERENCES [dbo].[User] ([ID])

);
```

Таблица 3. Comment

```
CREATE TABLE [dbo].[Comment] (

    [ID]          INT            IDENTITY (1, 1) NOT NULL,

    [AdvertiseID] INT            NOT NULL,

    [Anonymous]   BIT            NOT NULL,

    [CommentText] NVARCHAR (MAX) NULL,

    [CommentTime] DATETIME2 (7)  NOT NULL,

    [Commenter]   NVARCHAR (MAX) NULL,

    CONSTRAINT [PK_Comment] PRIMARY KEY CLUSTERED ([ID] ASC),

    CONSTRAINT [FK_Comment_Advertise_AdvertiseID] FOREIGN KEY
([AdvertiseID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE

);
```

Таблица 4. Compliment

```
CREATE TABLE [dbo].[Compliment] (

    [ID]           INT            IDENTITY (1, 1) NOT NULL,

    [AdvertiseID]  INT            NOT NULL,

    [Cleanness]    INT            NOT NULL,

    [Comfort]      INT            NOT NULL,

    [PriceQuality] INT            NOT NULL,

    [Reviewer]     NVARCHAR (MAX) NULL,

    [Staff]        INT            NOT NULL,

    CONSTRAINT [PK_Compliment] PRIMARY KEY CLUSTERED ([ID] ASC),

    CONSTRAINT [FK_Compliment_Advertise_AdvertiseID] FOREIGN KEY
([AdvertiseID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE

);
```

Таблица 5. Image

```
CREATE TABLE [dbo].[Image] (

    [ID]          INT             IDENTITY (1, 1) NOT NULL,

    [AdvertiseID] INT             NOT NULL,

    [FlatImage]   VARBINARY (MAX) NULL,
```

```
    CONSTRAINT [PK_Image] PRIMARY KEY CLUSTERED ([ID] ASC),

    CONSTRAINT [FK_Image_Advertise_AdvertiseID] FOREIGN KEY
([AdvertiseID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE

);
```

Таблица 6. RentRage

```
CREATE TABLE [dbo].[RentRage] (

    [ID]          INT           IDENTITY (1, 1) NOT NULL,

    [AdvertiseID] INT           NOT NULL,

    [RentFrom]    DATETIME2 (7) NOT NULL,

    [RentTo]      DATETIME2 (7) NOT NULL,

    [Status]      TINYINT       NOT NULL,

    CONSTRAINT [PK_RentRage] PRIMARY KEY CLUSTERED ([ID] ASC),

    CONSTRAINT [FK_RentRage_Advertise_AdvertiseID] FOREIGN KEY
([AdvertiseID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE

);
```

Таблица 7. Review

```
CREATE TABLE [dbo].[Review] (

    [ID]          INT            IDENTITY (1, 1) NOT NULL,

    [AdvertiseID] INT            NOT NULL,

    [ReviewStar]  INT            NOT NULL,

    [Reviewer]    NVARCHAR (MAX) NULL,

    CONSTRAINT [PK_Review] PRIMARY KEY CLUSTERED ([ID] ASC),

    CONSTRAINT [FK_Review_Advertise_AdvertiseID] FOREIGN KEY
([AdvertiseID]) REFERENCES [dbo].[Advertise] ([ID]) ON DELETE CASCADE

);
```

Таблица 8. User

```
CREATE TABLE [dbo].[User] (

    [ID]       INT             IDENTITY (1, 1) NOT NULL,

    [Address]  NVARCHAR (MAX)  NOT NULL,

    [Avatar]   VARBINARY (MAX) NULL,

    [Contact]  NVARCHAR (MAX)  NOT NULL,

    [Email]    NVARCHAR (450)  NOT NULL,

    [Name]     NVARCHAR (MAX)  NOT NULL,
```

```
    [Password] NVARCHAR (MAX)  NOT NULL,
    [Role]     NVARCHAR (MAX)  NULL,
    CONSTRAINT [PK_User] PRIMARY KEY CLUSTERED ([ID] ASC),
    CONSTRAINT [AlternateKey_Email] UNIQUE NONCLUSTERED ([Email] ASC)
);
```

3. Реализация приложения

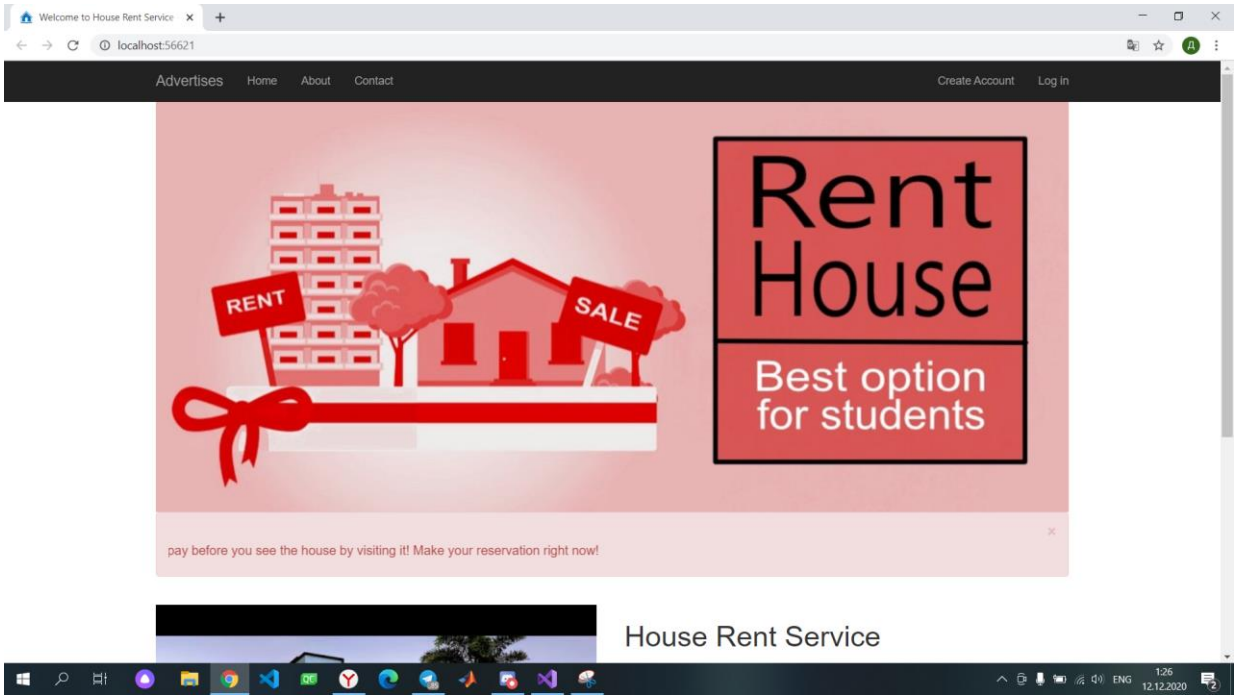


Рисунок 1. Стартовая страница сайта (ч.1)

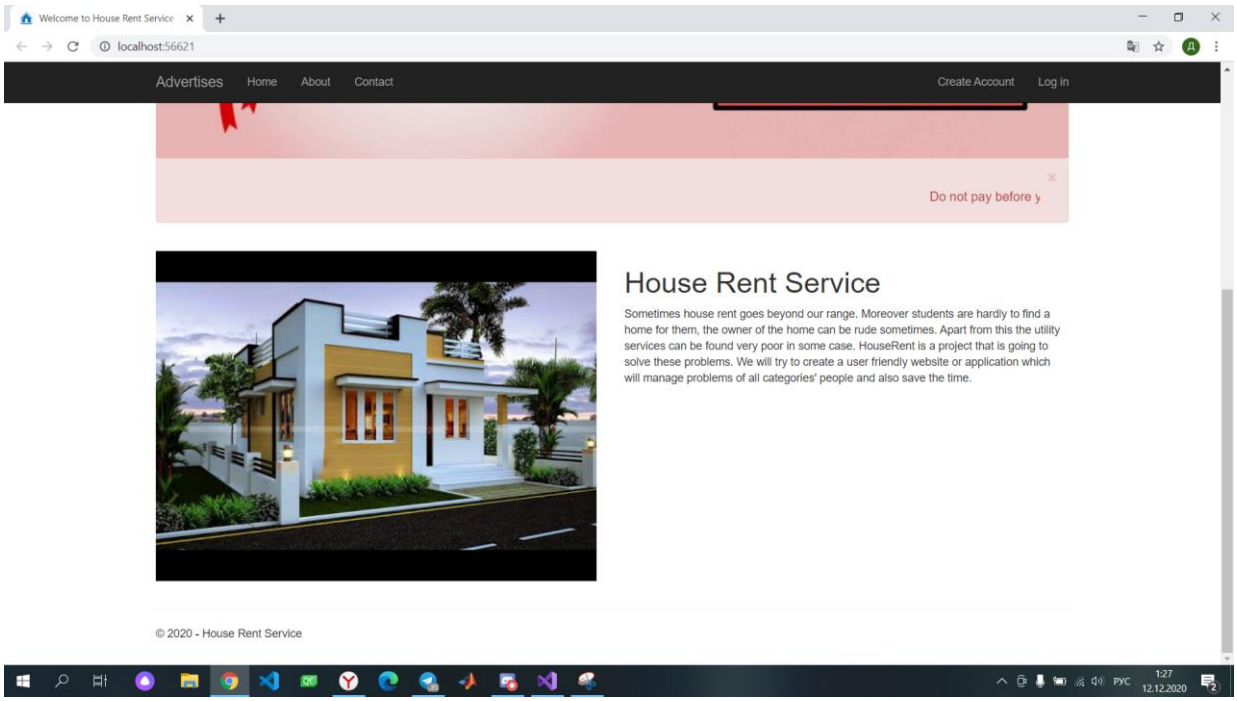


Рисунок 2. Стартовая страница сайта (ч.2)

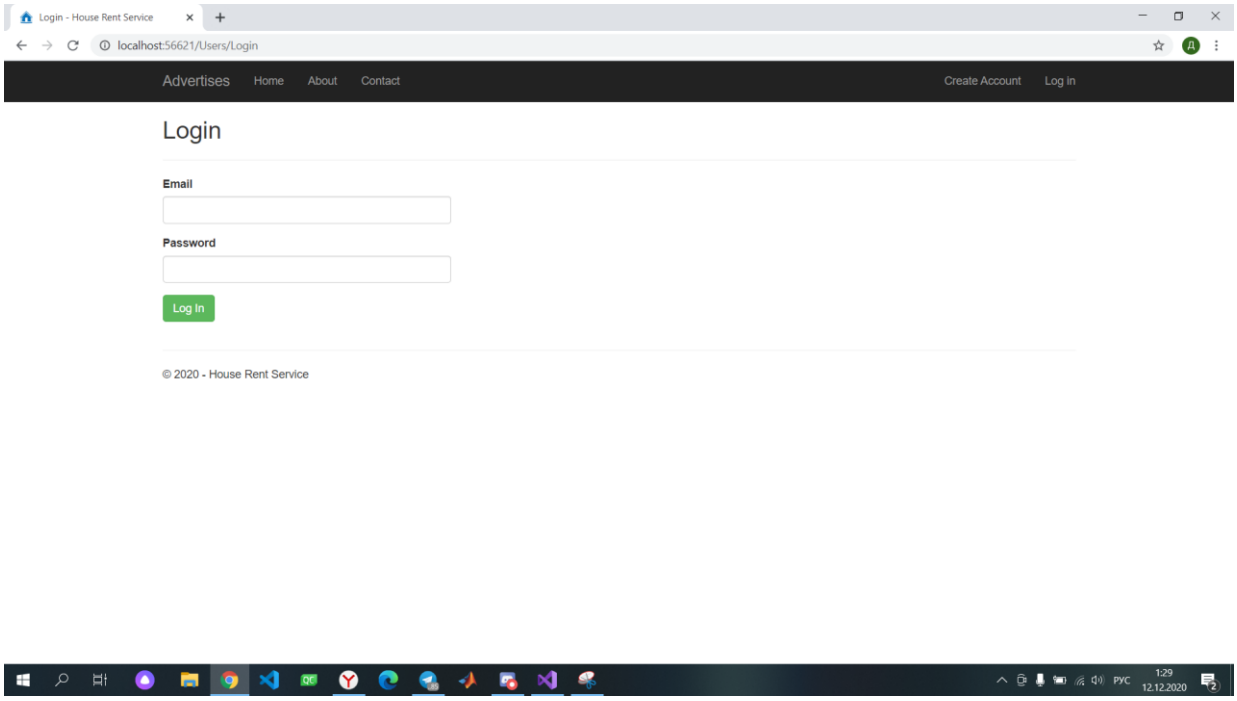


Рисунок 3. Страница входа

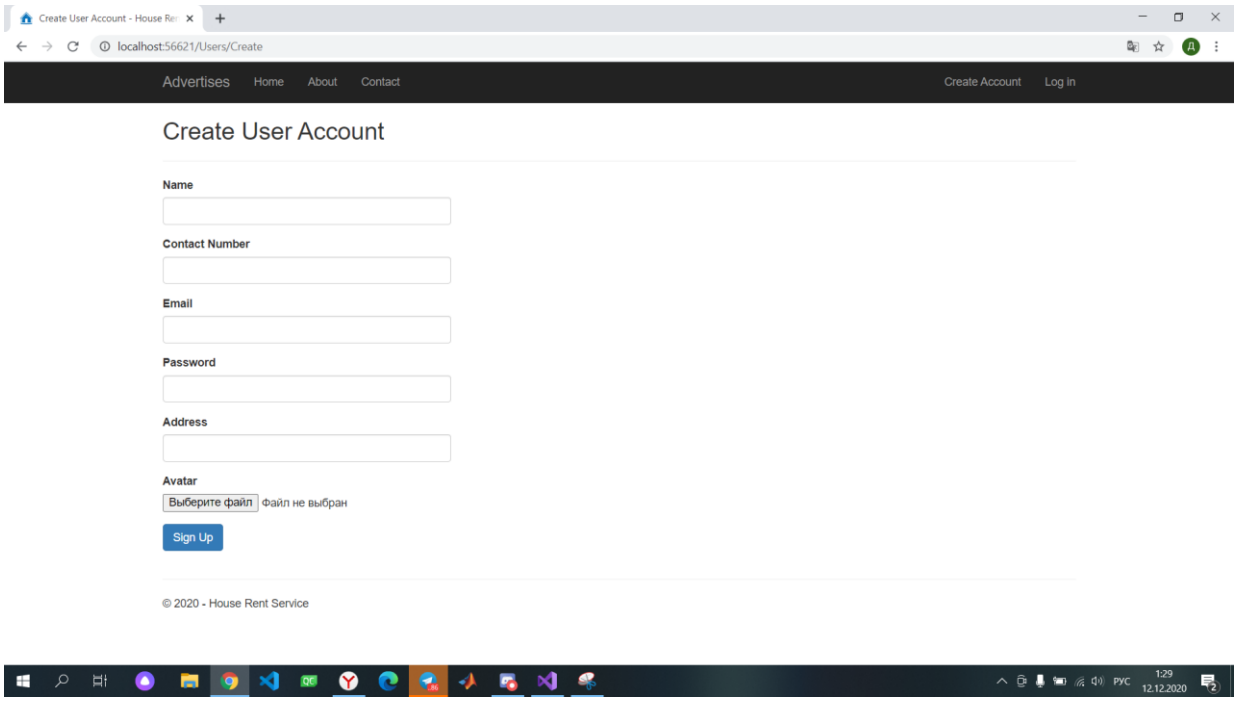


Рисунок 4. Страница регистрации

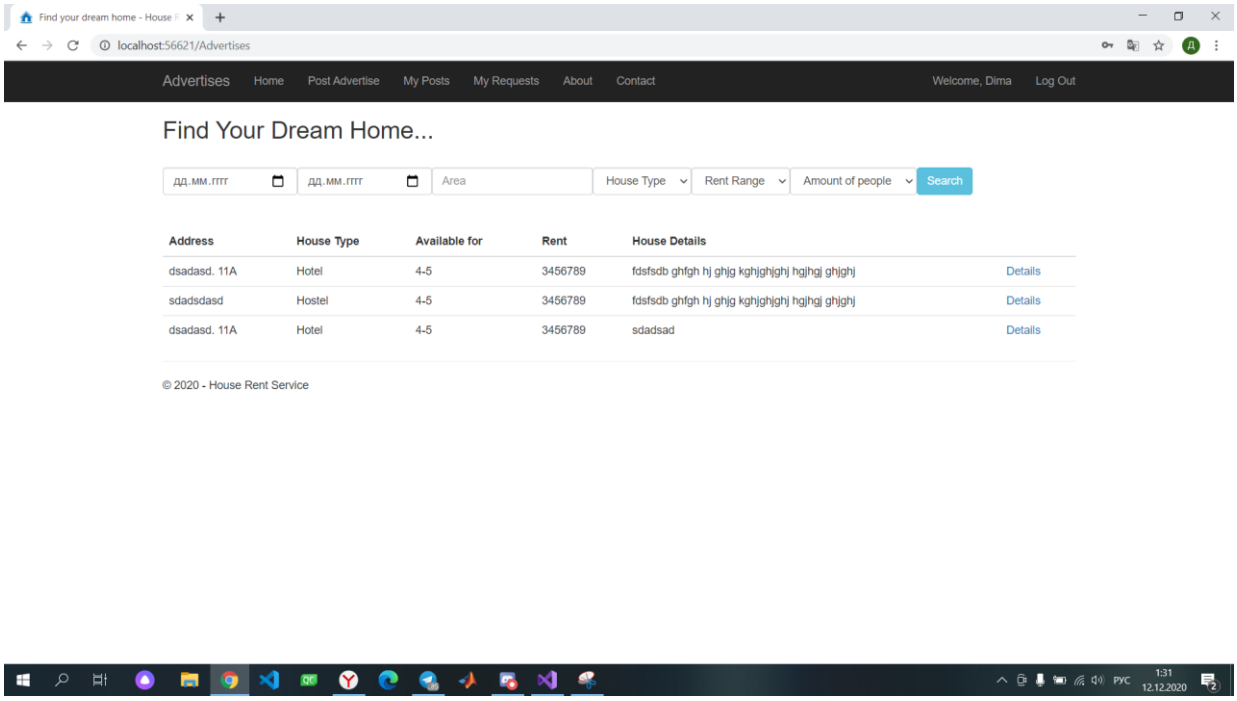


Рисунок 5. Страница всех предложений аренды

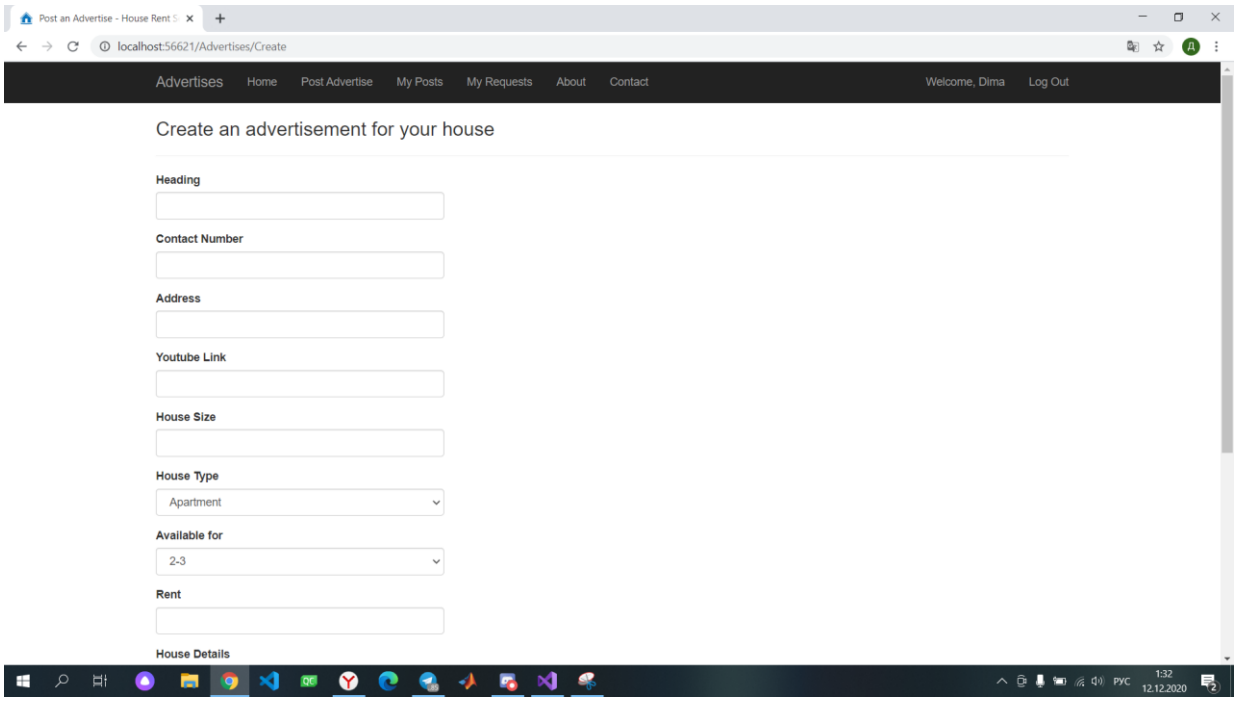


Рисунок 6. Страница создания публикации аренды

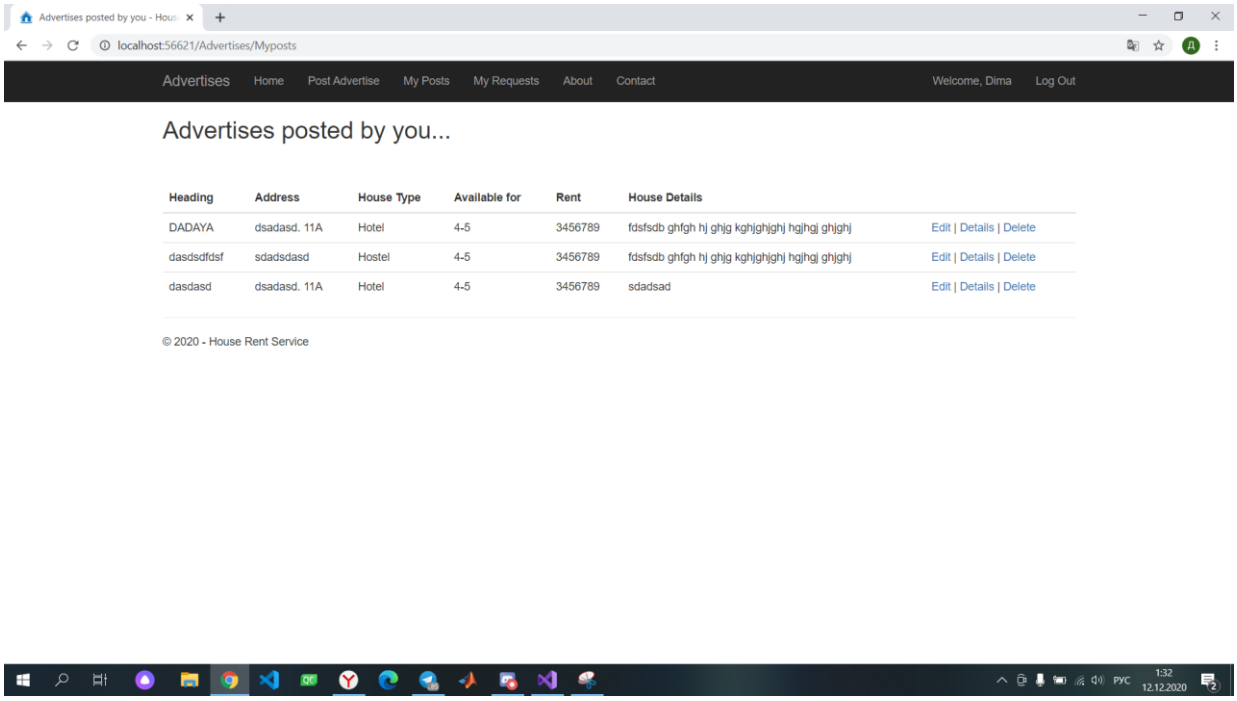


Рисунок 7. Страница публикаций пользователя

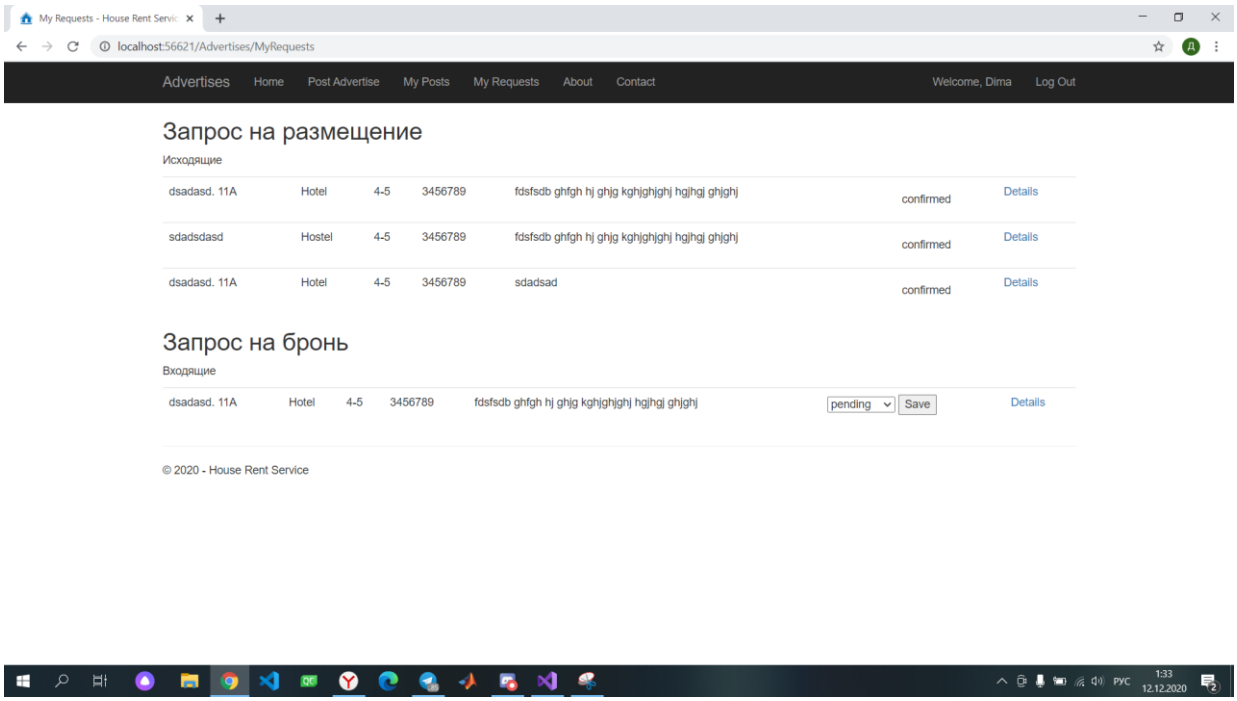


Рисунок 8. Страница запросов на бронь/публикацию пользователя

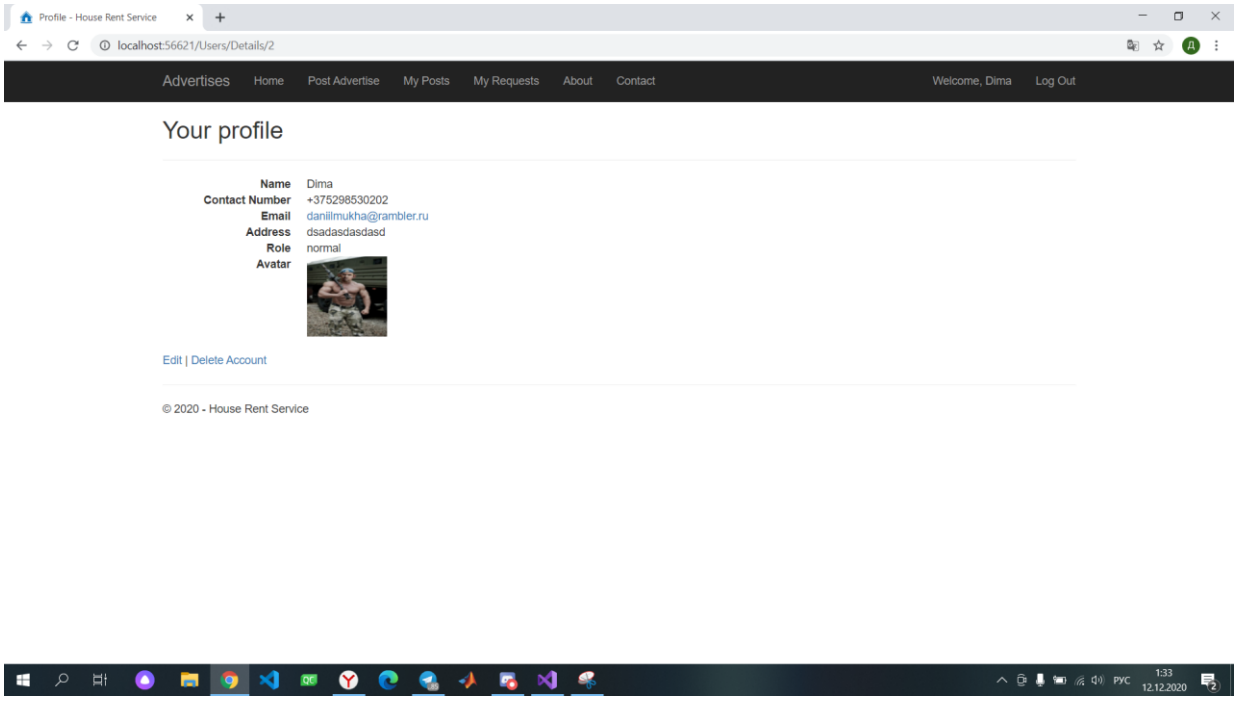


Рисунок 9. Страница профиля пользователя

Заключение

В ходе работы с базой данных mysql был разработан сайт для отображения данных базы, который позволяет арендовать и выставлять на аренду временное жилье.

Список используемых источников

ASP.NET Core documentation - https://metanit.com/sharp/aspnet5/29.1.php

RazorPages documentation - https://www.learnrazorpages.com/

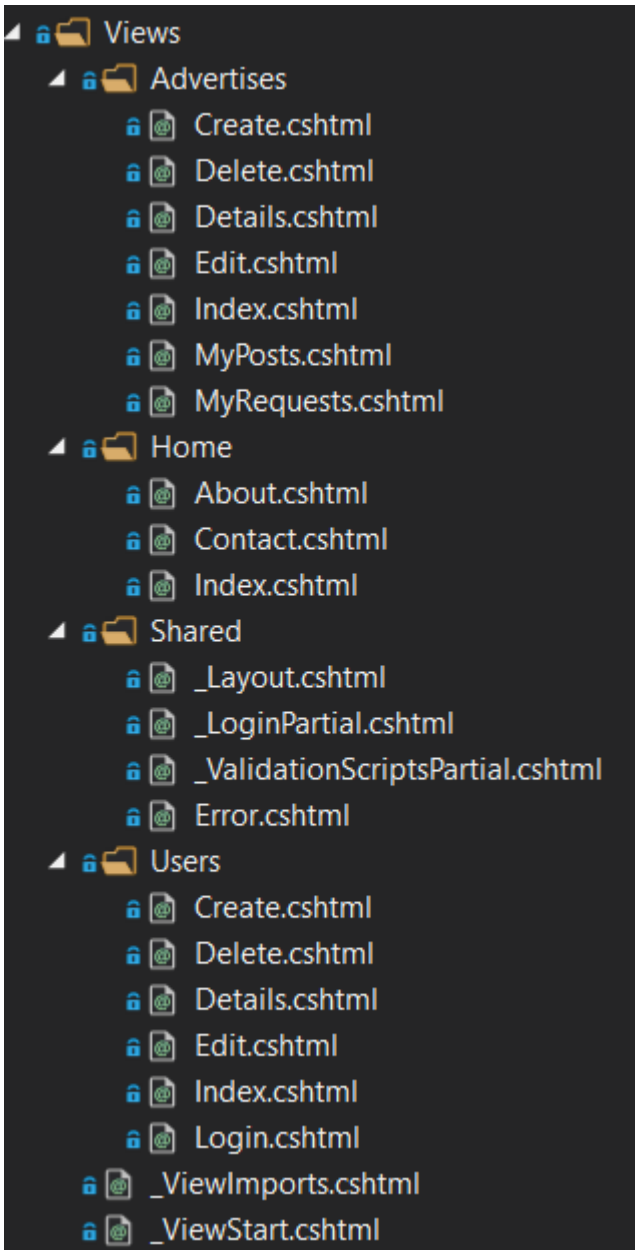
Entity Framework documentation - https://www.learnentityframeworkcore.com/

C# documentation - https://docs.microsoft.com/ru-ru/dotnet/csharp/

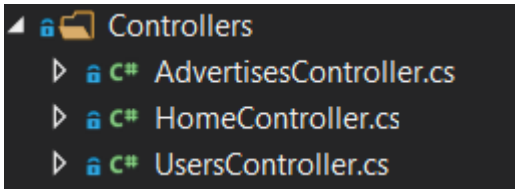
Приложение 1. Код программы

Код и структура проекта - https://github.com/Muha113/house-rent

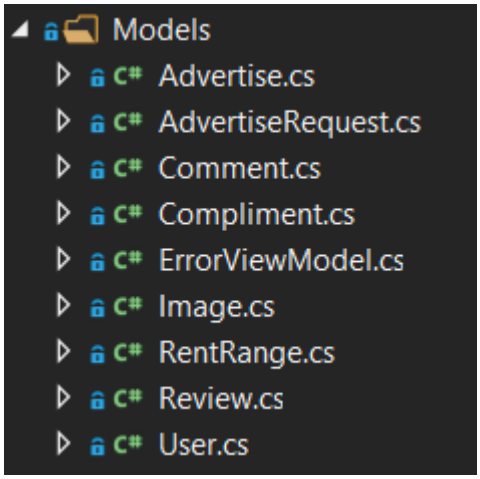
Views



Controllers



Models



Contollers/HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using HouseRent.Models;
using Microsoft.AspNetCore.Http;

namespace HouseRent.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult About()
        {
            ViewData["Message"] = "Your application description page.";
```

```
            return View();
        }


        public IActionResult Contact()
        {
            ViewData["Message"] = "Your contact page.";


            return View();
        }




        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }
    }
}
```

Contollers/UserController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using HouseRent.Models;
using System.IO;
using Microsoft.AspNetCore.Http;
using HouseRent.Services;
using Microsoft.Extensions.Configuration;
```

```
namespace HouseRent.Controllers

{

    public class UsersController : Controller

    {

        private const string userString =
"ID,Name,Contact,Email,Password,Address,Role,Avatar";

        private readonly HouseRentContext _context;

        private readonly EmailService _emailService;

        private readonly IConfiguration _config;



        public UsersController(HouseRentContext context, EmailService emailService,
IConfiguration config)

        {

            _context = context;

            _emailService = emailService;

            _config = config;

        }



        //A funtion to return image path from database

        public FileContentResult GetImg(int id)

        {

            var image = _context.User.Find(id).Avatar;



            return image != null ? new FileContentResult(image, "image/png") : null;

        }



        // GET: Users

        public async Task<IActionResult> Index()

        {

            //this means only admin can see the user list

            if(HttpContext.Session.GetString("sRole") != "admin")

            {

                return RedirectToAction("Index", "Home");
```

```
            }
            return View(await _context.User.ToListAsync());
        }



        public IActionResult Login()
        {
            //this means u cannot reload login page once u logged in
            if(!String.IsNullOrEmpty(HttpContext.Session.GetString("sEmail")))
            {
                return RedirectToAction("Index", "Home");
            }
            return View();
        }


        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Login([Bind(userString)] User user)
        {
            var usr = await _context.User
                        .SingleOrDefaultAsync(u => u.Email.ToUpper() ==
user.Email.ToUpper()
                        && u.Password == user.Password);
            if (usr == null)
            {
                HttpContext.Session.SetString("loginFailed", "Email & Password didn't
matched!");
                return View();
            }
            HttpContext.Session.SetString("sName", usr.Name);
            HttpContext.Session.SetString("sEmail", usr.Email);
            HttpContext.Session.SetString("sRole", usr.Role);
            HttpContext.Session.SetString("sId", usr.ID.ToString());
```

```
    HttpContext.Session.Remove("loginFailed");

    return RedirectToAction("Index", "Advertises");

}

public IActionResult Logout()
{
    //all session variables, sets as empty...
    HttpContext.Session.Remove("sName");
    HttpContext.Session.Remove("sEmail");
    HttpContext.Session.Remove("sRole");
    HttpContext.Session.Remove("sId");
    HttpContext.Session.Remove("loginFailed");
    HttpContext.Session.Remove("userExist");

    return RedirectToAction(nameof(Login));
}

// GET: Users/Details/5
public async Task<IActionResult> Details(int? id)
{
    //this means u cannot load details page without logged in
    if (String.IsNullOrEmpty(HttpContext.Session.GetString("sEmail")))
    {
        return RedirectToAction("Index", "Home");
    }
    if (id == null)
    {
        return NotFound();
    }
```

```
    var user = await _context.User
        .SingleOrDefaultAsync(m => m.ID == id);
    if (user == null)
    {
        return NotFound();
    }
    else
    {
        //this means admins and owners only can see their account details
        if(user.Email != HttpContext.Session.GetString("sEmail")
            && HttpContext.Session.GetString("sRole") != "admin")
        {
            return RedirectToAction("Index", "Home");
        }
    }

    return View(user);
}

// GET: Users/Create
public IActionResult Create()
{
    //this means u cannot load create/signup page once u logged in
    if (!String.IsNullOrEmpty(HttpContext.Session.GetString("sEmail")))
    {
        return RedirectToAction("Index", "Home");
    }
    return View();
}

// POST: Users/Create  (Used for sign up)
[HttpPost]
```

```
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind(userString)] User user, IFormFile
img)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    //for image : if given an image or not
                    if(img != null)
                    {
                        if (img.Length > 0)
                        {
                            using (var ms = new MemoryStream())
                            {
                                img.CopyTo(ms);
                                var fileBytes = ms.ToArray();
                                user.Avatar = fileBytes;
                            }
                        }
                    }

                    user.Role = "normal"; //others user are either admin or banned
                    _context.Add(user);
                    await _context.SaveChangesAsync();
                    HttpContext.Session.SetString("sName", user.Name);
                    HttpContext.Session.SetString("sEmail", user.Email);
                    HttpContext.Session.SetString("sRole", user.Role);
                    HttpContext.Session.SetString("sId", user.ID.ToString());
                    HttpContext.Session.Remove("userExist");
                    var task = Task.Run(async () =>
                    {
```

```
                        using (var es = _emailService.SendEmailAsync(user.Email, "You
have been registered in HouseRent.", $"Login: {user.Email}\nPassword: {user.Password}"))

                        {

                            await es;

                        }

                    });

                    return RedirectToAction(nameof(Index));

                }

                catch

                {

                    HttpContext.Session.SetString("userExist", user.Email+" Already
Exist. Go to Login Page.");

                    return View(user);

                }

            }

            return View(user);

        }


        // GET: Users/Edit/5

        public async Task<IActionResult> Edit(int? id)

        {

            //this means u cannot load edit page without logged in

            if (String.IsNullOrEmpty(HttpContext.Session.GetString("sEmail")))

            {

                return RedirectToAction("Index", "Home");

            }

            if (id == null)

            {

                return NotFound();

            }


            var user = await _context.User.SingleOrDefaultAsync(m => m.ID == id);

            if (user == null)
```

```
            {
                return NotFound();
            }
            else
            {
                //this means owners only can edit their account details
                if (user.Email != HttpContext.Session.GetString("sEmail"))
                {
                    return RedirectToAction("Index", "Home");
                }
            }
            return View(user);
        }

        // POST: Users/Edit/5
        // To protect from overposting attacks, please enable the specific properties you want to bind to, for
        // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind(userString)] User user)
        {
            if (id != user.ID)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(user);
                    await _context.SaveChangesAsync();
```

```
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!UserExists(user.ID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    return View(user);
}

// GET: Users/Delete/5
public async Task<IActionResult> Delete(int? id)
{
    //this means u cannot load delete page without logged in
    if (String.IsNullOrEmpty(HttpContext.Session.GetString("sEmail")))
    {
        return RedirectToAction("Index", "Home");
    }
    if (id == null)
    {
        return NotFound();
    }

    var user = await _context.User
        .SingleOrDefaultAsync(m => m.ID == id);
```

```
    if (user == null)
    {
        return NotFound();
    }
    else
    {
        //this means admins and owners can delete an account
        if (user.Email != HttpContext.Session.GetString("sEmail")
            && HttpContext.Session.GetString("sRole") != "admin")
        {
            return RedirectToAction("Index", "Home");
        }
    }

    return View(user);
}

// POST: Users/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var user = await _context.User.SingleOrDefaultAsync(m => m.ID == id);
    _context.User.Remove(user);
    await _context.SaveChangesAsync();

    //if the owner deletes his account, he will be logged out
    if(HttpContext.Session.GetString("sEmail") == user.Email)
    {
        return RedirectToAction(nameof(Logout));
    }
    return RedirectToAction(nameof(Index));
```

```
        }


        private bool UserExists(int id)

        {

            return _context.User.Any(e => e.ID == id);

        }

    }

}
```