|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Студент |  | | /Алексеев Степан Владимирович / | | | |
|  |  | |  | | | |
| Факультет | ИВТ | | Группа | | ИП-712 |  |
| Руководитель |  | | /Рубан Анатолий Альбертович/ | | | |
|  |  |  | |  | | |
|  | |  | |  | | |
|  |  | |  | | | |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ПЕРЕВОДА ДЕНЕЖНЫХ СРЕДСТВ МЕЖДУ БАНКОВСКИМИ КАРТАМИ

Пояснительная записка

Кафедра ПМиК

Допустить к защите

зав. кафедрой: проф., д.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фионов А.Н.

**Министерство цифрового развития, связи и массовых коммуникаций РФ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Новосибирск 2021 г.

**Министерство цифрового развития, связи и массовых коммуникаций РФ**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Форма утверждена

научно-методическим советом

ФГОБУ ВПО «СибГУТИ»

протокол № 2 от 04.03. 2014 г.

**КАФЕДРА Прикладной математики и кибернетики**

**ЗАДАНИЕ**

**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

|  |  |
| --- | --- |
| СТУДЕНТУ Алексееву С.В. | ГРУППЫ ИП-712 |

УТВЕРЖДАЮ

\_\_\_\_\_\_\_\_\_\_2021

зав. кафедрой ПМиК

\_\_\_\_\_\_\_\_\_\_\_\_\_**/** Фионов А.Н. /

Новосибирск 2021 г.

**1. Тема выпускной квалификационной работы бакалавра**

Разработка веб-приложения для перевода денежных средств между

банковскими картами

утверждена приказом СибГУТИ от «19» января 2021 г. № 4/37о-21

**2. Срок сдачи студентом законченной работы** «12» июня 2021 г.

**3. Исходные данные к работе**

1. Asp.net: на сайте youtube [Электронный ресурс] URL: https://www.youtube.com/watch?v=U4nNZS1bnj8&list=PL0lO\_mIqDDFWltIe7D6aUS5f4k1y2-rgn&ab\_channel=%D0%93%D0%BE%D1%88%D0%B0%D0%94%D1%83%D0%B4%D0%B0%D1%80%D1%8C Гриффитс Д., Гриффитс Д. Head First. Программирование для Android / пер. с англ. 2-е изд. СПб. : Питер, 2018. 912 с.
2. Asp.net: официальная документация [Электронный ресурс] URL: https://dotnet.microsoft.com/learn/aspnet

|  |  |
| --- | --- |
| **4. Содержание пояснительной записки**  **(перечень подлежащих разработке вопросов)** | **Сроки выполнения по разделам** |
| 1. Введение | 01.02.2021-19.02.2021 |
| 1. Постановка задачи | 20.02.2021-26.02.2021 |
| 1. Выбор средств разработки | 27.02.2021-05.03.2021 |
| 1. Программная реализация | 06.03.2021-24.04.2021 |
| 1. Руководство пользователя | 25.04.2021-03.05.2021 |
| 1. Заключение, список использованных источников | 04.05.2021-30.05.2021 |

Дата выдачи задания «01» февраля 2021 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рубан А.А.

*подпись*

Задание принял к исполнению «20» мая 2021 г.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексеев С.В.

*подпись*

**Федеральное агентство связи**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

Форма утверждена научно-методическим

советом университета

протокол № 3 от 19.02.2015 г.

**ОТЗЫВ**

на выпускную квалификационную работу студента Алексеева Степана Владимировича

по теме «Разработка веб-приложения для перевода денежных средств между банковскими картами»

Оценка уровней сформированности общекультурных и профессиональных компетенций обучающегося:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Компетенции | | Уровень сформированности  компетенции | | |
| Высокий | Средний | Низкий |
| Общекультурные | ОК-7 |  |  |  |
| Профессиональные | ОПК-4 |  |  |  |
| ОПК-5 |  |  |  |
| ПК-1 |  |  |  |
| ПК-2 |  |  |  |
| ПК-3 |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Работа имеет практическую ценность |  | Тема предложена предприятием |  |
| Работа внедрена |  | Тема предложена студентом |  |
| Рекомендую работу к внедрению |  | Тема является фундаментальной |  |
| Рекомендую работу к опубликованию |  | Рекомендую студента в магистратуру |  |
| Работа выполнена с применением ЭВМ |  | Рекомендую студента в аспирантуру |  |

Руководитель бакалаврской работы доцент кафедры ПМиК, к.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Рубан Анатолий Альбертович, 10.06.2021

(должность, уч. степень, подпись, фамилия, имя, отчество (полностью), дата)

Пояснения к кодам компетенций (не подшивать!)

ОК-7 способность к самоорганизации и самообразованию

ОПК-4 способность участвовать в настройке и наладке программно-аппаратных комплексов

ОПК-5 способность решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности

ПК-1 способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели и интерфейсов "человек – электронно-вычислительная машина"

ПК-2 способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования

ПК-3 способностью обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности

**АННОТАЦИЯ**

Выпускной квалификационной работы студента Алексеева С.В.

по теме «Разработка веб-приложения для перевода денежных средств между банковскими картами»

Объём работы - 0 страниц, на которых размещены -1 рисунков и -1 листингов. При написании работы использовалось 19 источников.

Ключевые слова: ASP, .NET, SQL, C#, база данных, веб-приложение

Работа выполнена на кафедре прикладной математики и кибернетики (ПМиК) СибГУТИ.

Руководитель доцент кафедры ПМиК, к.т.н., доцент Рубан Анатолий Альбертович.

Основные результаты: создано веб-приложение для перевода денежных средств между банковскими картами с помощью фреймворка ASP.NET. В процессе создания приложения изучен язык программирования C# и освоена среда разработки Visual Studio.

Оглавление

[1 Введение 3](#_Toc74302300)

[2 Постановка задачи 5](#_Toc74302301)

[2.1 Исследование предметной области 5](#_Toc74302302)

[2.2 Поиск аналогов и прототипов 5](#_Toc74302303)

[2.3 Цели и задачи разработки 7](#_Toc74302304)

[2.4 Выбор средств разработки 7](#_Toc74302305)

[2.4.1 Среда разработки Visual Studio 7](#_Toc74302306)

[2.4.2 Язык C# 7](#_Toc74302307)

[2.4.3 Фреймворк ASP.net 7](#_Toc74302308)

[2.4.4 Фреймворк Identity.net 8](#_Toc74302309)

[3 ПРограммная реализация 12](#_Toc74302310)

[3.1 Окна 12](#_Toc74302311)

[3.2 Структура приложения 17](#_Toc74302312)

[3.3 Реализация базы данных 20](#_Toc74302313)

[3.4 Этапы разработки 21](#_Toc74302314)

[3.4.1 Создание проекта 21](#_Toc74302315)

[3.4.2 Создание таблиц в БД 22](#_Toc74302316)

[3.4.3 Создание страниц 23](#_Toc74302317)

[3.4.4 Создание контроллеров 25](#_Toc74302318)

[4 Заключение 27](#_Toc74302319)

[5 список использованных источников 28](#_Toc74302320)

[6 Приложение А 29](#_Toc74302321)

# Введение

Безналичнй расчёт является итогом эволюции товарно-денежных отношений. Изначально, очевидно, людьми использовался бартер, когда ценная вещь/услуга обменивались на вещь/услугу той же ценности. Со временем такими предметами становились более лёгкие по весу и объёму предметы(жемчуг, шкуры животных, золото, серебро). Такая ситуация продолжалась и до недавнего времени, когда считалось, что валюта(бумажные деньги) должна быть поддержана золотым запасом выпускающей эти деньги страны. Однако в итоге имеем ситуацию, когда такого количества золота просто нет. При этом надёжность валюты всё же можно гарантировать обязательствами, выраженными документально. Здесь государство уже рискует просто разрушением своей экнономики, если не будет обеспечивать ценность своих денег.

Сегодня мы имеем возможность расплачиваться электронными банковскими картами, мобильными телефонами с фукнцией NFC, имитирующими ту же банковскую карту. Так же в последнее время начала распространение технология расчёта по идентификации по лицу. Сбербанк предлагает такую услугу. Для расчёта у кассы нужно просто посмотреть в камеру. Если лицо распознается и станет ясно, что в банке такой клиент зарегистрирован, то произойдёт списане денежных средств с его счёта.

Возвращаясь к эволюции денег приходим к тому, что вес и объём средства расчётов приравнялся к нулю в случае расчёта посредством распознаванием по лицу.

Я в своей работе постарался изучить расчёты банковскими картами в интернете и реализовать их в виде веб-приложения для браузера.

Банковская карта привязывается к специальному карточному счёту в банке. Далее при возможности пользоваться онлайн(при наличии веб-сайта или приложения) владелец карты может совершать покупки в интернет магазинах, пересылать деньги другим людям на другие карты в том числе в другие банки. При этом требуется подтверждение в виде ввода пароля из смс, присланного банком или подтверждения в приложении. Большинство людей, которых я знаю пользуются такими приложениями в своих смартфонах. Также многие совершают покупки в магазинах через компьютер через веб-браузер. Там требуется указать номер карты, имя владельца и другую информацию при покупке. В таком случае уже веб-браузер посылает сообщение на номер телефона или в приложение на телефоне для получения подтверждения от владельца карты в соответствии с законом о национальной платёжной системе.

В большинстве стран с развитой экономикой пластиковые карты участвуют в торговле наравне с деньгами. В таких странах более девяноста процентов операций с деньгами проводится в электронном виде.

Изготавливаются карты из пластиковой основы и микрочипа или магнитной ленты. На лицевой стороне обычно шестнадцати или восемнадцатизначный номер карты, имя владельца, дата истечения срока обслуживания. На обратной стороне трёхзначный код CVI для подтверждения некоторых операций.

Обычно процентная ставка на карточном счёте гораздо ниже, чем на обычном, поэтому такие счета менее привлекательны. Но удобство побеждает.

В мобильном приложении, как и в веб-приложении, можно отследить все свои операции для предотвращения случаев хищения и мошенничества и просто для ведения личной бухгалтерии. Есть возможность настроить автоматическую оплату коммунальных услуг, услуг связи и т.п.

Рынок банковских услуг в сфере интернет-банкинга в последнее время сильно увеличился, поэтому необходимость в спецалистах, работающих с такими системами возросла. Вследствие этого мною и была выбрана такая тема для ВКР.

В пояснительной записке описаны элементы разработанного мной приложения. Это база данных банка, таблицы, пользовательский интерфейс, классы, контроллеры. Больше всего посвящено описанию программы.

# Постановка задачи

## Исследование предметной области

Существует большое количество различных банковских систем, например ABS, систем первода денег, например VISA, МИР. Многие из них ргламентированы международными правовыми актами и государственными законами. В России это закон о национальной платёжной системе. Всё,

## Поиск аналогов и прототипов

Существует множество уже реализованных систем, подобно разрабатываемой мною. Например сбербанк онлайн:

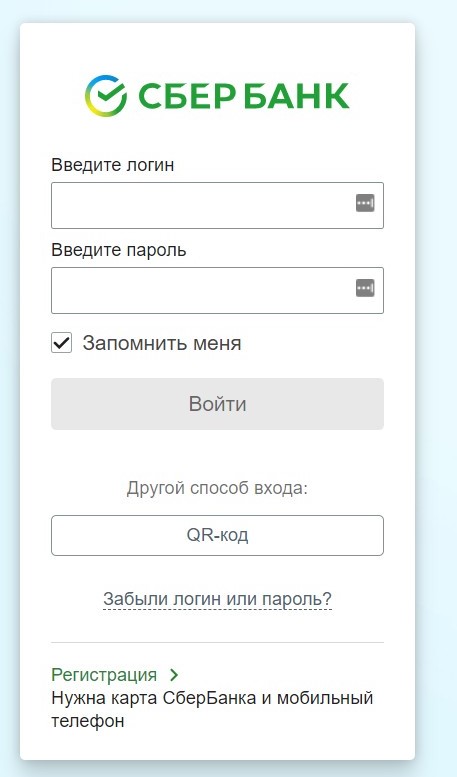


Рисунок 2.1

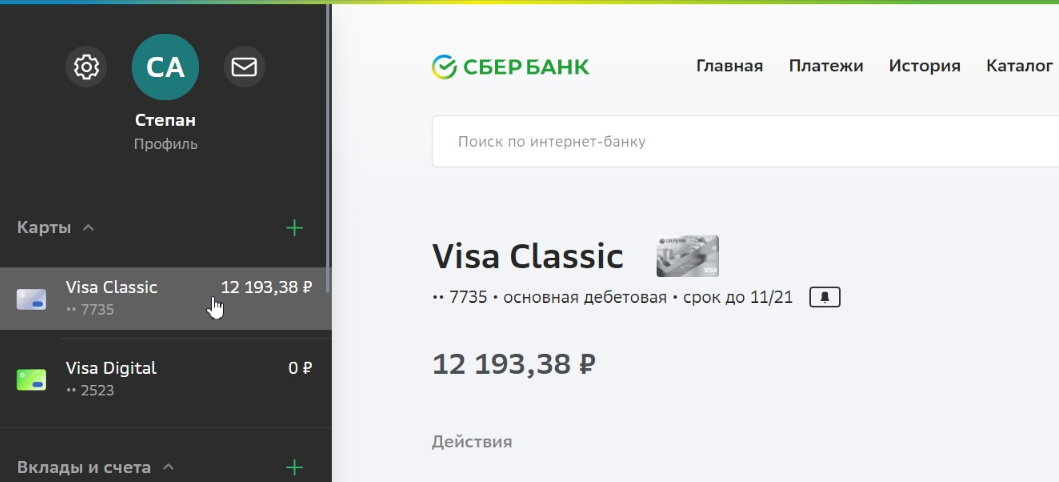


Рисунок 2.2

На рисунке 2.2 видны все карты, принадлежащие клиенту.

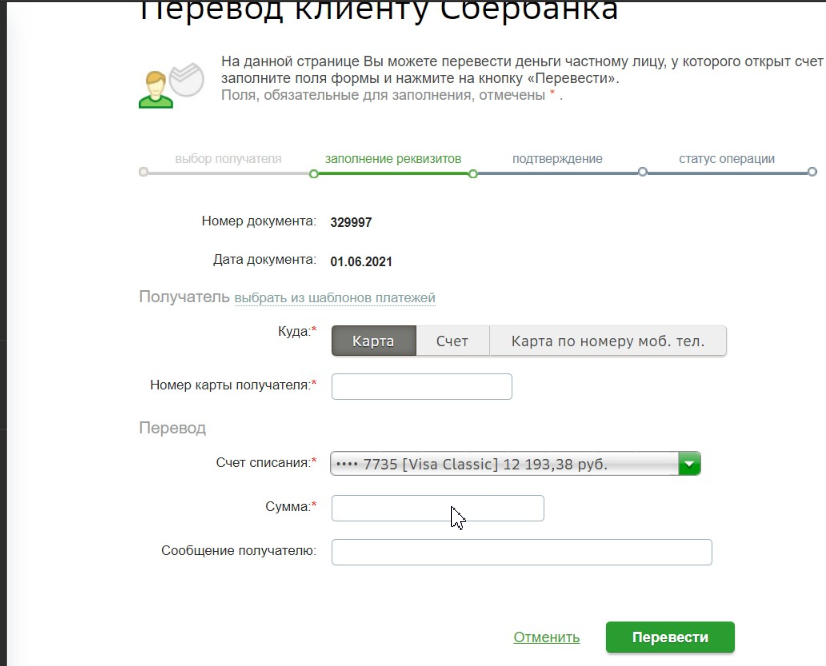


Рисунок 2.3

На рисунке 2.3 представлено окно для перевода денег.

В принципе видно, что выглядит всё понятно и просто. Сначала зарегистрироваться, выбрать одну из своих карт, потом ввести реквизиты получателя и перевести деньги.

## Цели и задачи разработки

Во-первых требуется создать интерфейс пользователя. Нужно добиться запрещения доступа к элементам, не принадлежащим незарегистрированным пользователям. Т.е. доступ по лоигну и паролю.

Во-вторых нужно разработать базу данных, хранящую информацию о пользователях, их картах, счетах, истории транзакций.

В-третьих нужно разработать бэк-енд приложение, реализующее логику переводов денег.

## Выбор средств разработки

В качестве средств разработки были выбраны:

1. Среда разработки Visual Studio

2. Язык C#

3. Фреймворк asp.net

4. Фреймворк Identity.net

### Среда разработки Visual Studio

Данный продукт компании Microsoft предназначен для разработки приложений на разных языках программирования. В основном – это язык C#. В Visual Studio можно создавать приложения с помощью разных популярных фреймворков. Очень удобно наличие хорошо структурированной, подробной документации. Мною была выбрана среда разработки Visual Studio, т.к. мне нравится наличие хорошо написанной документации для фреймворков компании Майкрософт.

### Язык C#

Данный язык является си-подобным, что следует из названия. Разработан компанией Microsoft для поддержки объектно-ориентированного программирования. Также он является компонентно-ориентированным, что значит, что кроме объектов предполагается управление и наборами объектов, модулями, независимыми друг от друга, способными работать по отдельности. Также предполагается наличие управления разными процессами. Создаются экземпляры и используются компоненты, также как объекты, как внутри одного процесса, так и между разными процессами.

### Фреймворк ASP.net

Хотелось изучить современный фреймворк с хорошей документацией, поэтому решил выбрать ASP.NET, т.к. у них много обучающих видео, стройная структура кода, фреймворки сделаны в одной организации в одном стиле, что урощает их изучение. Данный фреймворк может использоваться для создания множества типов приложений. Я буду использовать ту часть, которая предназначена для веб-разработки. Будут использоваться Razor Pages – файлы типа .cshtml(похожие на HTML файлы с кодом на языке C#).

### Фреймворк Identity.net

Это фреймворк для работы с аутентификацией.

Возможны два подхода для реализации:

1 автоматическая регистрация. Можно сделать на сайте Azure portal page for app registration.

2 ручная настройка.

Части программ общаются друг с другом с помощью токенов security token service. Identity был создан(несмотря на наличия множества уже существующих решений) для того, чтобы придать гибкости приложению, чтобы пользователи могли выполнять больше действия так, как им хочется. Это централизованное решение. Имеется в виду, что пользователь входит один раз, получает токен и вносит нужные ему изменения.

Identity server доставляется с помощью NuGet пакета. Ответ на вопрос, как Identity связывается с ПО в том, что он преимущественно реализуется как промежуточное ПО. Идея в том, что ты создаёшь хост с этим Identity Server, заходишь в NuGet, прикрепляешь промежуточное ПО в очередь запроса(pipeline). В итоге работа данного промежуточного ПО(Identity Server) – добавить конечные точки (endpoints) для ранее упоминавшихся протоколов(open ID connect, Oauth2). Т.е. идея в том, что я не беру своё приложение и добавляю в него Identity Server, а в том, что я делаю отдельный Хост, т.к. реализую я STS(Security Token Service, Сервис токенов для безопасности). Т.е это новое веб-приложение, работающее как бы параллельно с моим основным приложением.

Для этого кстати есть шаблоны. Токены – это то, что мы доставляем назад в приложения для доказательства аутентичности пользователя и токены – это то, что приложение будет использовать для обращения к API. Токен содержит информацию о пользователе и он также подписан, т.е. есть цифровая подпись, созданная с помощью STS. В этом ценность открытых систем, в возможности использовать подобные настройки(управление временем сессии пользователя и другие вещи). А точнее получается, что Identity Server – это фреймворк, позволяющий построить свою STS. Т.е. это не просто готовая STS, это также набор инструментов для построения своей собственной STS. Т.е. об Identity Server можно говорить как о фреймворке, это библиотека, которую подключают и строят новое приложение со своим поведением. Это поведение в итоге соотвтетсвует нужным разработчикам требованиям.

Если сравнивать с другими подобными решениями, то для Java, например популярна Pink Federate. Azure Active Directory – облачное STS, предлягаемая компанией Майкрософт. Также есть OCTA, AUTH0. Т.е. можно создать свой хост, а можно доверить это облачным сервисам.

В итоге мы имеем централизованное управление с одиночным вхождением пользователя в систему(и возможностью проводить множество операций после этого вхождения).

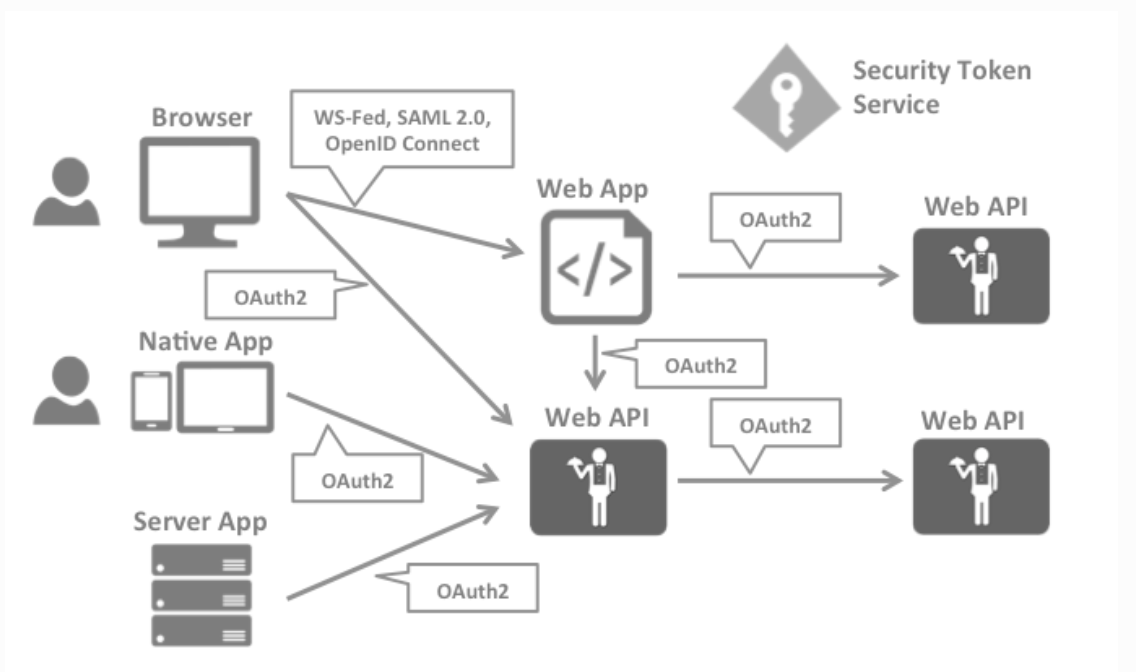


Рисунок 2.4

Т.о. в итоге мы будем управлять сервисом безопасности с токенами. Web-api и web-app, которые я создам будут общаться друг с другом посредством этого сервиса. Всё это будет работать в соответсвии с Open Id connect & Oauth2 protocol. Красота решения в том, что Identity можно использовать в программе, но если потребуется, то можно будет его сменить на подобный сервис. Identity server будет сервисом для использования токенов.

В Visual Studio вероятно в будущем, можно будет зайти в файл – новый проект – Identity Server. Т.о. есть промежуточное ПО(middleware), реализующее эти конечные точки протоколов. Далее наш клиент в виде приложения типа Model View Controller(приложение, желающее знать, кем является наш пользователь) будет говорить через протокол с Identity Server.

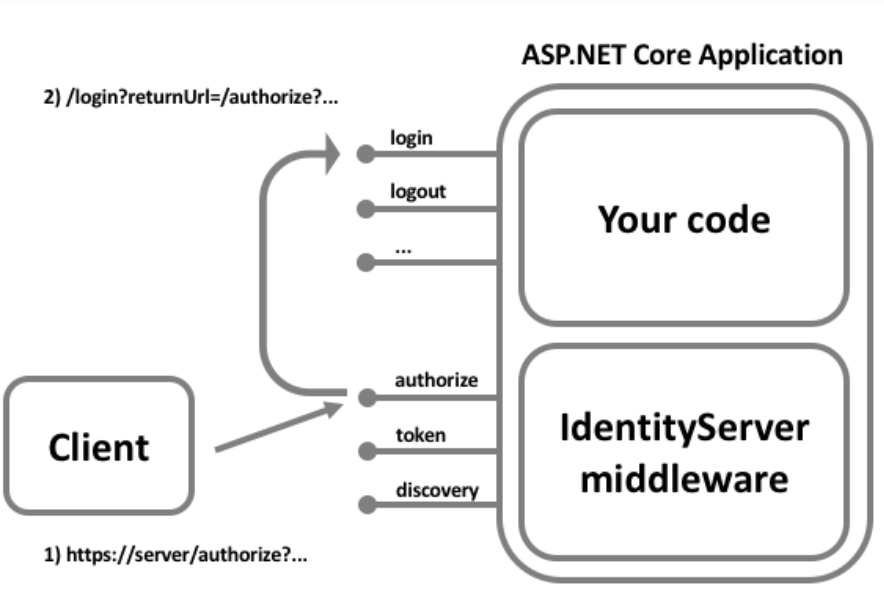


Рисунок 2.5

Также можно использовать и другое ПО для авторизации. На рисунке клиент шлёт запрос на авторизацию, Identity Server пересылает его на настроенную пользователем страничку авторизации. Клиент вводит все необходимые данные. Потом ты опять пересылаешь клиента(пользователя) на конечную точку авторизации(authorize endpoint) в нижней части картинки. Конечная точка авторизации – это протокол конечной точки, где пользователю показывается пользовательский интерфейс.

Можно действовать так: создаю отдельное приложение в проекте, прописываю в нём в Startup.cs в методе Configure app.UseIdentityServer(); после app.UseStaticFiles(); при этом Identity Server подключён как NuGet package. Пока Identity Server не знает ничего про наших клиентов и пользователей. Его надо научить этому.

В целом asp.net Identity – это библиотека или фреймворк, созданная для управления базой данных, содержащей информацию об аутентификации пользователей. Эта информация в первую очередь: пароль, почта, имя. Также там много функциональности реализовано, такой как подтверждение почты, двухфакторная аутентификация, подтверждение номера мобильного телефона и т.п. Также удобно думать об asp.net как о библиотеке, говорящей с базой данных, содержащей наших пользователей. Однако Identity не реализует протоколы безопасности. app.UseIdentityServer также добавляет ПО для работы с аутентификацией посредством файлов куки в Хост приложение. Однако можно и самому реализовать эту функциональность.

Половина Identity Server настраивается с помощью системы внедрения зависимостей. Identity Server предоставляет набор сервисов, которые доступны для внедрения как зависимости. Для подключения этих сервисов нужно в классе Startup.cs, в методе ConfigureServices добавить services.AddIdentityServer(); это сконфигурирует сервисы ядра. После этого нужно будет вызвать некоторые расширяющие методы, чтобы настроить нужную функциональность.

Identity Server очень гибкий и был сконструирован для того, чтобы быть расширяемым. Поэтому нужны конфигурационные данные, нужные IdSe. Причём нет строгого требования по тому, где хранить эти данные. Поэтому список MVC приложений, список API может быть в базе данных, может быть в файле JSON или можно даже его прописать в виде кода в приложении.

# ПРограммная реализация

## Окна

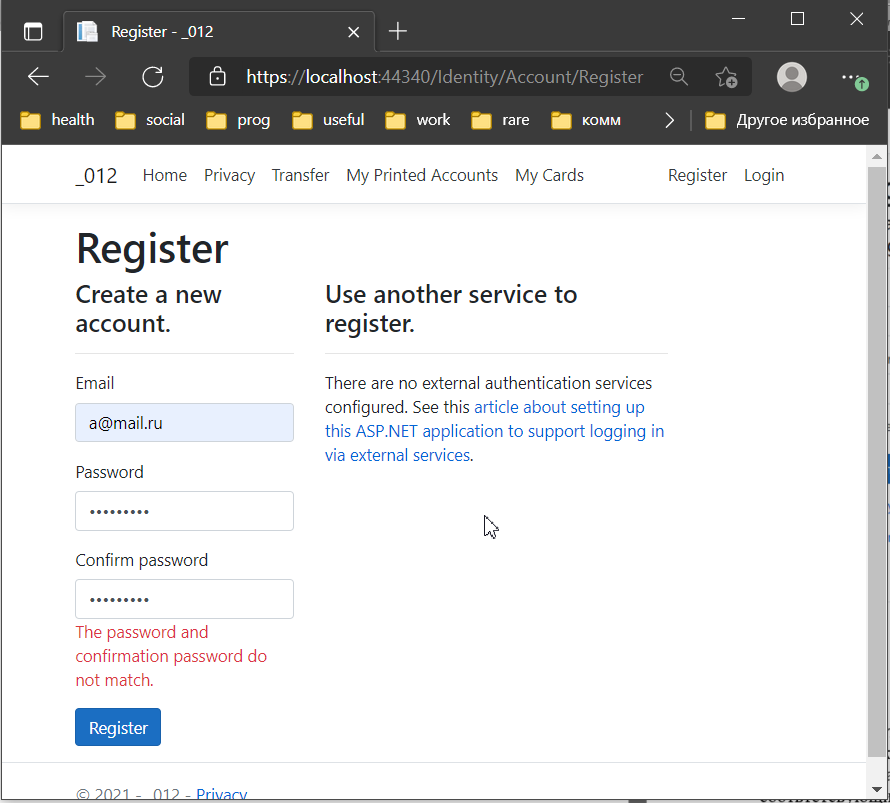


Рисунок 3.1

При первом вхождении в программу нужно зарегистрироваться. В пароле должны быть маленькие и большие латинские буквы, цифры и небуквенные символы, такие как точки, запятые и т.п. После регистрации нужно подтвердить почту. Упрощенно это делается нажатием на согласие прямо в текущем окне. После этого можно войти, нажатием на кнопку Login и введя пароль. До ввода пароля все кнопки меню ведут на страницу авторизации.

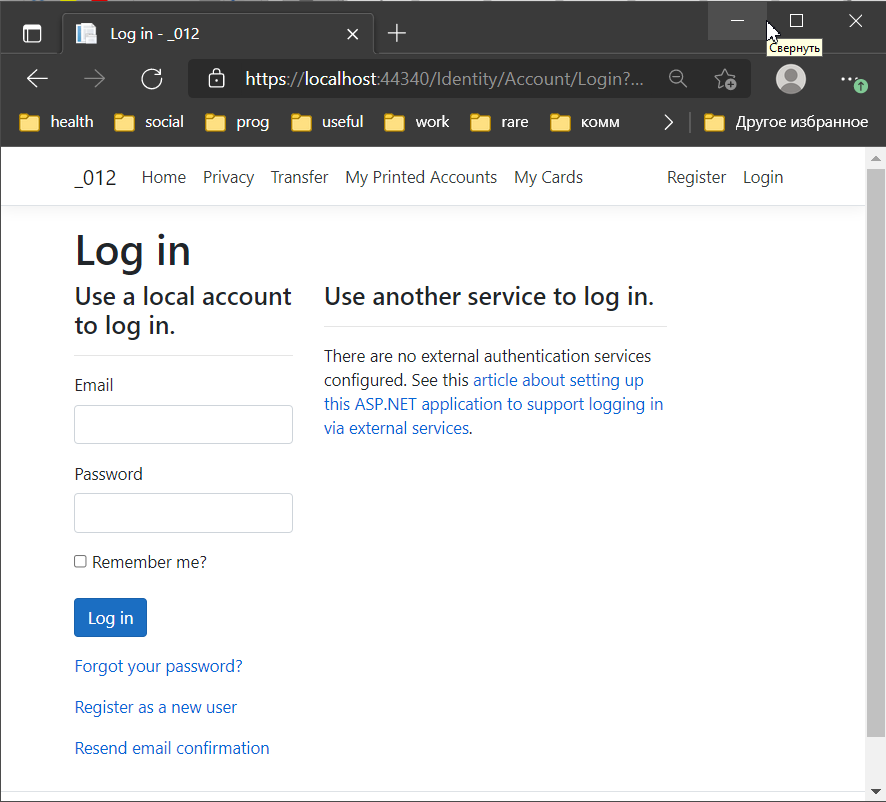


Рисунок 3.2

Доступ к кнопкам меню закрыт для незарегистрированных пользователей.

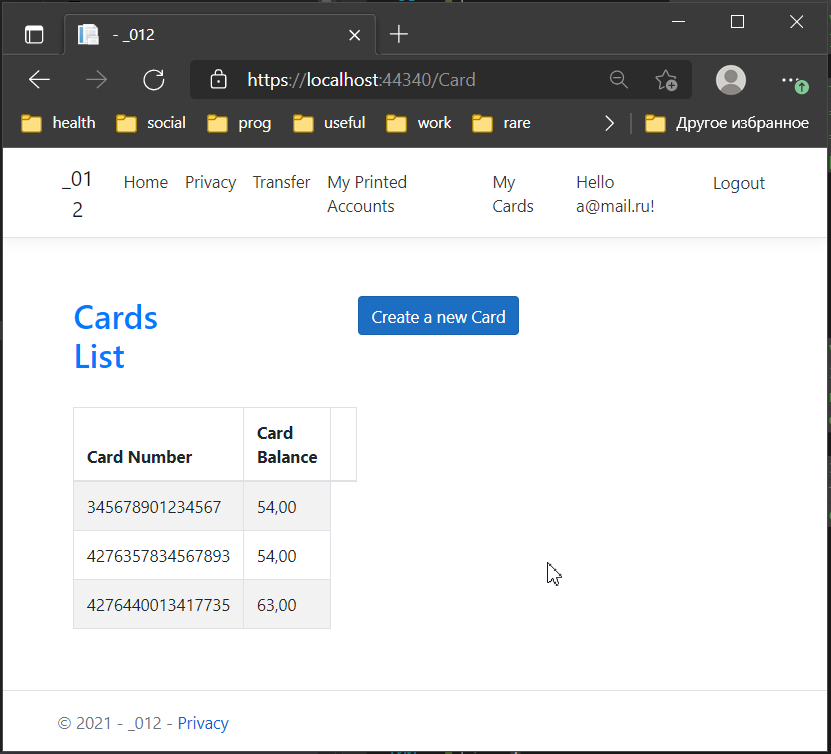


Рисунок 3.3

После авторизации в разделе карт можно увидеть все свои карты или создать новую.

В адресной строке подразумевается конечно, 44340/Card/Index, но Index опускается.

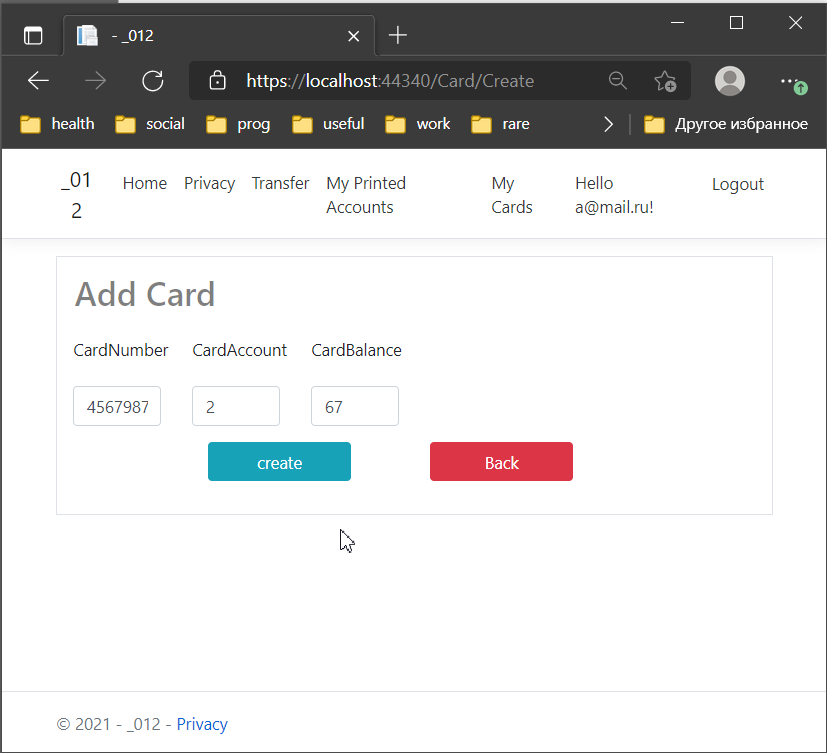


Рисунок 3.4

Как видно, отличие между адресами в адресной строке на рисунках 3.3 и 3.4 в последних словах Index и Create. Эти слова определяют и вызываеют соответствующие методы в контроллере CardController. В итоге создаётся новая карта с введёнными данными. Далее она хранится в таблице CardT.

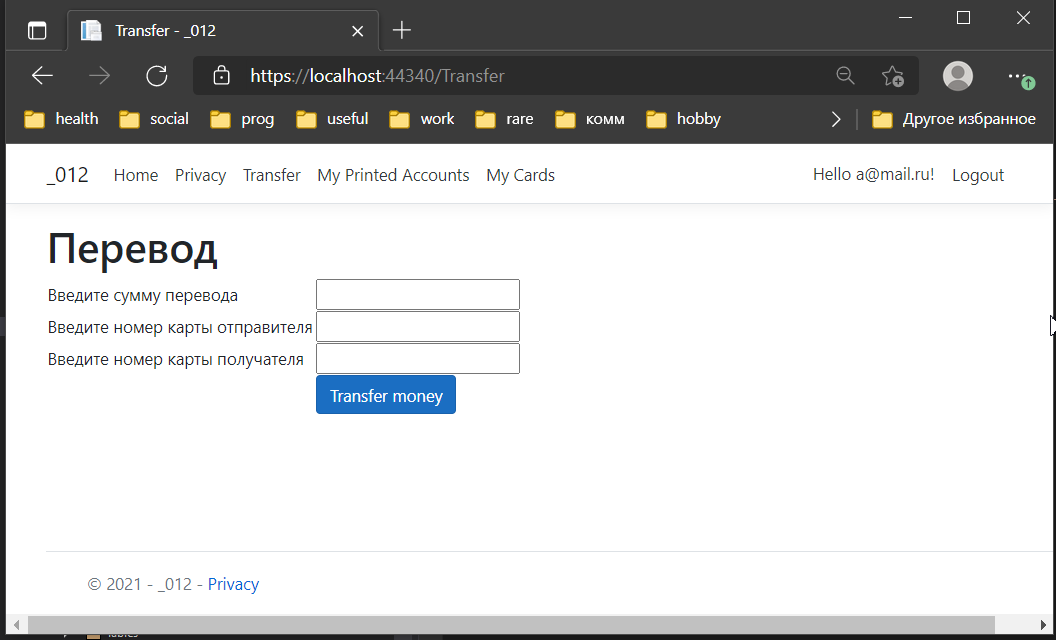


Рисунок 3.5

В окне перевода выбирается карта отправителя, получателя и сумма перевода. По нажатию кнопки Transfer money совершается проверка наличия достаточного количества денег, наличие ограничений на переводы и, в случае успешности проверок, происходит перевод.

## Структура приложения

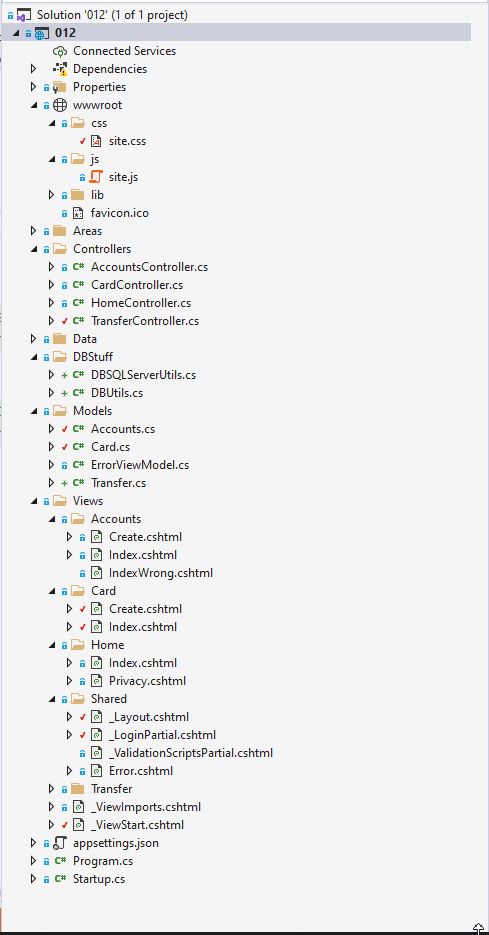


Рисунок 3.6

В структуре приложения можно увидеть все составные части. Connected Services показывает поключённые базы данных, даёт возможность выбрать способ публикации приложения, нарпимер с помощью Azure или Docker или с помощью веб-сервера.

В зависимостях(Dependencies) можно увидеть подключённые зависимости, фреймворки, ПО-посредников:

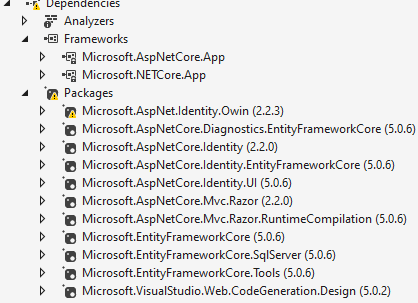


Рисунок 3.7

В wwwroot видна статическая информация, такая как site.css, определяющий, соответсвенно стиль всего сайта. Также в site.js можно написать код на JavaScript.

В Controolers указаны все созданные мной контроллеры, управляющие странцами, прописанными в первой части названий файлов, например CardController.cs – контроллер, управляющий страницами(View), расположенными в папке Card.

Data содержит информацию о всех пройденных миграциях(перестройке базы данных в соотвтетствии с изменённой мной конфигурацией в файлах папки Models. Таким образом, с помощью этой информации можно отследить все изменения базы данных. Это своеобразная система контроля версий.

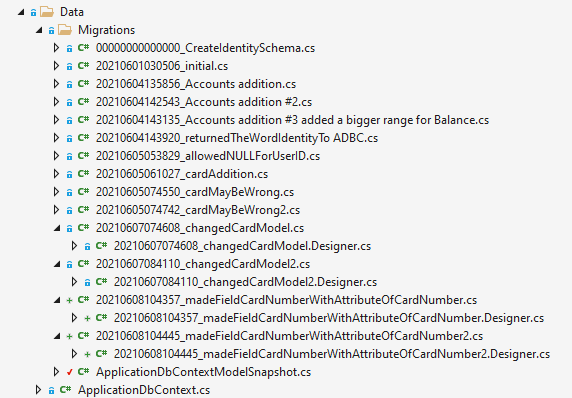


Рисунок 3.8

DBStuff содержит экспериментальный код для подключения к базе данных.

Models содержит информацию о таблицах, которые я вносил в базу данных. Файлы, содержащиеся там описывают строение базы данных, которое потом используется при миграциях.

Views содержит файлы и папки для непосредственной разметки страниц с помощью кода HTML, а также, в случае страниц Razor для написания дополнительного кода на языке C#.

\_ViewImports.cshtml позволяет увидеть импортированные библиотеки.

\_ViewStart.cshtml позволяет настроить страницу, с которой начнётся выполнение приложения.

Appsettings.json участвует в подключении к базе данных. Там можно прописать имя базы данных, её тип, адрес. В итоге к ней будет совершено подключение, либо её создание, если её не было раньше.

Program.cs создаёт приложение(хост), вызывая методы файла Startup.cs.

Startup.cs настраивает конфигурацию, поключая промежуточное ПО из зависимостей, включает все конфигурационные настройки.

3.1 Общее описание работы приложения

При старте приложения нужно зарегистрироваться в окне регистрации или войти по уже зарегистрированным логину и паролю.

При нажатии кнопок My Cards, My Accounts, Transfers можно увидеть соотвтетсвующие разделы, созданные посредством взаимодействия с базой данных. При нажатии Transfers можно перейти к переводу денег с карты на карту.

## Реализация базы данных

База данных создана с помощью миграций из встроенных в фреймворк Identity моделей, а также из моделей, созданных мной и лежащих в папке Models. Например Card.cs:

public class Card

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int Id { get; set; }

[Column(TypeName = "decimal(16)")]

[CreditCard]

public decimal CardNumber { get; set; }

public string CardAccount { get; set; }

[Column(TypeName = "decimal(18,2)")]

public decimal CardBalance { get; set; }

public string CardHolderId { get; set; }

public string CardHolderName { get; set; }

public DateTime expiryDate { get; set; }

public int cvi { get; set; }

[Column(TypeName = "decimal(18,2)")]

public decimal changeBalance { get; set; }

}

Здесь указано, что поле ID – первичный ключ и он добавляется, обновляется(инкрементируется базой данных). Поле CardNumber помечено, как объект, хранящий номер электронной банковской карты с помощью [CreditCard]. Для поля баланса указан диапазон допустимых значений в 18 знаков целых чисел и два после запятой.

Так таблица, содержащая информацию о картах CardT выглядит в среде разработки:

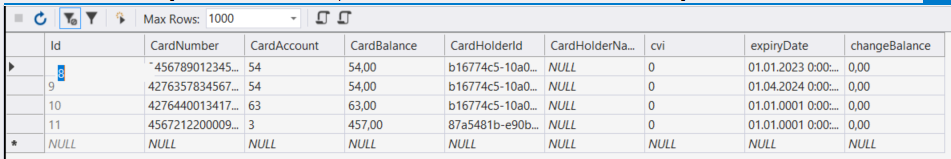


Рисунок 3.9

## Этапы разработки

### Создание проекта

Сначала создаётся приложение ASP MVC. Это создаёт приложение с подключенным промежуточным ПО для работы в виде модель – вид – контроллер. Это видно по соответсвующим появляющимся в проектк папкам Controllers, Models, Views.

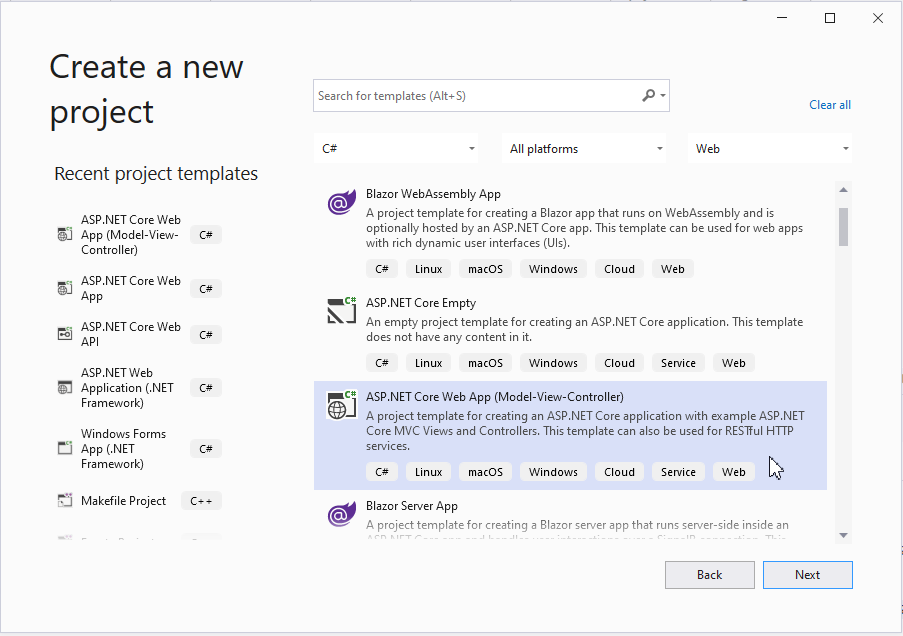


Рисунок 3.10

Выбирается тип приложения с индивидуальными аккаунтами. Таким образом автоматически создастся база данных пользователей и все миграции для неё будут видны. Ставится галочка Enable Razor runtime compilation для того, чтобы при обновлении содержимого файлов Razor не нужно было перезапускать весь проект, а достаточно было просто обновить страницу в браузере.

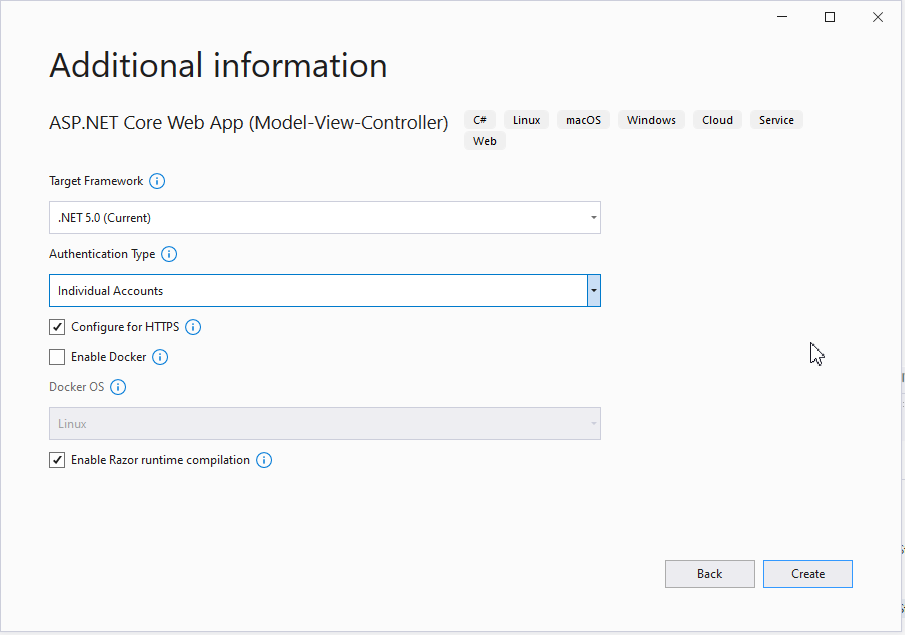


Рисунок 3.11

### Создание таблиц в БД

После создания приложения я добавил в БД таблицы для карт, аккаунтов, переводов.

Добавление таблиц делается путём так называемых миграций, когда разработчиком создаётся модель, например для таблицы перевода создаётся файл Transfer.cs в папке Models со следующим содержимым:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace \_012.Models

{

public class Transfer

{

public decimal cardFrom { get; set; }

public decimal cardTo { get; set; }

public decimal amt { get; set; }

public DateTime date { get; set; }

}

}

Здесь каждая переменная является как бы заготовкой для столбца будущей таблицы. Для каждой переменной объявляется тип хранимой в ней информации. При чём в виде C# кода, а не в виде SQL типов столбцов, т.е. например, decimal, а не NUMBER. Это так называемая технология Object Relational Mapping, когда объекты языка программирования ставновятся объектами в БД. После этого в Package Manager Console прописывается команда add-migartion “описание внесённых в модель изменений”. Visual Studio создаёт файлы миграции, которые уже содержат в себе команды SQL. Эти команды создают подходящие таблицы в базе данных с преобразованными в SQL типами. Такой способ изменения БД подходит для небольших и не очень сложных проектов. Для крупных проектов с большой нагрузкой, конечно, лучше тщательнее контролировать БД, типы столбцов для лучшей производительности, и, миграции могут быть не лучшим решением. Т.е. предпочтение лучше отдать прямому взаимодействию с БД посредством команд SQL.

Для создания карт, привязанных к уникальным пользователям я использовал уникальные идентификаторы, присвоенные пользователям при создании таблицы dbo.AspNetUsers. Понятно, что это не лучшее решение, но для начала решение приемлемое, т.к. идентификаторы точно будут уникальными и будут являться ссылками на нужных пользователей. При создании карты этот идентификатор берётся у текущего пользователя и устанавливается в поле CardHolderId таблицы CardT. Таким образом я ограничиваю доступ пользователей только их картами, т.е. чужие карты они не смогут увидеть и поменять.

### Создание страниц

Помимо создания моделей я начал разработку соответствующих страниц, вносящих изменения в БД с помощью файлов-контроллеров. Например странца добавления карты Views/Card/Create.cshtml представлена в листинге:

@model \_012.Models.Card

@using Microsoft.AspNetCore.Identity

@inject SignInManager<IdentityUser> SignInManager

@inject UserManager<IdentityUser> UserManager

@{

var user = await UserManager.GetUserAsync(User);

}

<form method="post" **asp-action**="Create">

<div class="border p-3">

<div class="form-group row">

<h2 class="text-black-50 pl-3">Add Card</h2>

</div>

<div class="row">

<div class="col-12">

<div class="form-group row">

<div class="col-2">

<label **asp-for**="CardNumber"> </label>

</div>

<div class="col-2">

<label **asp-for**="CardAccount"></label>

</div>

<div class="col-2">

<label **asp-for**="CardBalance"></label>

</div>

<div class="col-2">

<!--<label asp-for=""></label>-->

</div>

</div>

<div class="form-group row">

<div class="col-2">

<input **asp-for**="CardNumber" class="form-control" />

</div>

<div class="col-2">

<input **asp-for**="CardAccount" class="form-control" />

</div>

<div class="col-2">

<input **asp-for**="CardBalance" class="form-control" />

</div>

<div class="col-2">

<!--<input asp-for="0" class="form-control" />-->

</div>

</div>

<div class="form-group row">

<div class="col-8 offset-2 row">

<div class="col">

<input type="submit" class="btn btn-info w-75" value="create" /><!-- width of 75-->

</div>

<div class="col">

<a **asp-action**="Index" class="btn btn-danger w-75">Back</a>

</div>

</div>

</div>

</div>

</div>

</div>

</form>

Здесь видно, как код на языке C# отделён от кода HTML с помощью символа @.

С помощью @inject внедрены зависимости. Это нужно для обеспечения слабой связанности объектов. Объекты в итоге взаимодейтсвуют друг с другом через абстракции. Это позволяет сделать программу более гибкой.

Здесь в <form method="post" **asp-action**="Create"> указывается, что форма будет срабатывать на вызов post, при этом вызывается функция Create контроллера CardController. Вызов этой формы происходит посредством нажатия на кнопку “Create a new Card” на странице Views/Card/Index.cshtml: <a **asp-controller**="Card" **asp-action**="Create" class="btn btn-primary">Create a new Card</a> Здесь в свою очередь указано, что имя контроллера начинается с Card, т.е. полное имя – CaedController, функция, вызываемая в контроллере – Create. Так же указан внешний вид кнопки – btn-ptimary.

Также на странице создания карты создается таблица с помощью тегов col-2 создаются столбцы. Максимальное число столбцов там 12 и число для каждой таблицы считается как 12 разделить на число после col, т.е. в данном случае будет 12/2 = 6 столбцов. С помощью тегов label создаются названия столбцов. В данном случае asp-for=”CardNumber” указывает, что для названия данного столбца будет использоваться имя столбца CardNumber из таблицы CardT. Указание на таблицу CardT же происходит с помощью @model \_012.Models.Card, т.е. в самой первой строчке листинга указывается к какой модели будет обращаться код с данной страницы.

Для ввода соотвтетсвенно используются уже не label, а input теги, но вот указание на столбец в таблице производится также: asp-for=”CardBalance”. По завершении создания новой карты перемещаемся обратно на страничку Index: <a **asp-action**="Index" class="btn btn-danger w-75">Back</a>

### Создание контроллеров

Для взаимодействия страниц, моделей и создания логики приложения используются контроллеры. Каждой странице назначается свой контроллер. Например CardController.cs управляет представлением Card.

Что происходит в контроллере:

private readonly ApplicationDbContext \_db;

public CardController(ApplicationDbContext db,

UserManager<IdentityUser> userManager,

SignInManager<IdentityUser> signInManager )

{

\_db = db;

\_userManager = userManager;

\_signInManager = signInManager;

}

На представленном фрашменте контроллера CardController представлено объявление переменной базы данных, а также приём переменной существующей базы данных в конструкторе. После присвоения -db = db можно пользоваться базой данных, переданной в контроллер. Эточасть внедрения зависимостей. Также внедряются зависимости userManager и signInManager для управления данными пользователей, подключенными с помощью фреймворка Identity.

Далее в контроллере определён метод, вызывающийся при переходе на страничку Index:

public IActionResult Index()

{

IEnumerable<Card> objList = \_db.CardT;

return View(objList);

}

Здесь в первой строчке тела метода копируем всю информацию из таблицы карт. Далее эту информацию передаём в представление. В представлении уже происходит выделение только тех карт, которые принадлежат данному пользователю.

Далее определён метод создания новой карты Create:

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Create(Card obj)

{

var user = \_userManager.GetUserAsync(User);

if(user.Result.Equals(null))return RedirectToAction("Index");

obj.CardHolderId = user.Result.Id.ToString();

\_db.CardT.Add(obj);

\_db.SaveChanges();

return RedirectToAction("Index");

}

Здесь [HttpPost] указывает, что этот метод вызовется только при вызове страницы, принимающей данные, что в свою очередь определяется в параметрах кнопки, вызывающей эту страницу(type="submit").

[ValidateAntiForgeryToken] позволяет производить более безопасные операции. Вставляет скрытое поле на форму. Его значение можно проверить в контроллере, который взаимодействет с этой формой. Работает незаметно, но позволяет убедиться, что пользователь на самом деле вошёл в систему и вводит данные с этой формы, а не откуда-то еще и не пытается её как-либо обойти. В общем эта функция добавляется очень легко, при этом резко увеличивая безопасность сайта. Для этого работает механизм токенов. В итоге при обращении к странице, программа проверяет, есть ли всё ещё токен у вошедшего в систему пользователя, является ли он всё ещё залогинившимся.

Далее проверяем, можем ли обратиться к уникальному иденитфикатору пользователя из таблицы вошедших пользователей. Если нет, то переходим назад на первую страницу с картами. Иначе создаём новую запись в таблице CardT(новую банковскую карту) с идентификатором пользователя из нашей системы, управляемой Identity.net. Переходим на страницу с картами.

# Заключение

Было разработано приложение с некоторым функционалом, похожим на тот, который есть в настоящих банках. Однако, это обычно секретная информация, поэтому вряд ли я смог бы сделать что-то применимое в реальности. Так что перспектив использования этого ПО нет. Представляет ценность только как способ изучения веб-программирования.

Были изучены фреймворки asp.net, Identity.net. Было изучено веб-программирование сайтов в парадигме .net.

Приложение работает так, как задумывалось с некоторыми недостатками вследствие новизны(для меня) технологии .net и веб-программирования в целом.

В принципе трудности, с которыми я столкнулся преодолимы. Нужно просто набить руку, понять все принципы MVC в контексте ASP.net, и следующие приложения будут делаться быстрее.

# список использованных источников

* + - 1. Asp.net: на сайте youtube [Электронный ресурс] URL: [https://www.youtube.com/watch?v=U4nNZS1bnj8&list=PL0lO\_mIqDDFWltIe7D6aUS5f4k1y2-rgn&ab\_channel=%D0%93%D0%BE%D1%88%D0%B0%D0%94%D1%83%D0%B4%D0%B0%D1%80%D1%8C](https://www.youtube.com/watch?v=U4nNZS1bnj8&list=PL0lO_mIqDDFWltIe7D6aUS5f4k1y2-rgn&ab_channel=%D0%93%D0%BE%D1%88%D0%B0%D0%94%D1%83%D0%B4%D0%B0%D1%80%D1%8C%20) (дата обращения: 12.04.2021).
      2. Библиотека СибГУТИ [Электронный ресурс] URL: <https://sibsutis.ru/lib/>(дата обращения 12.04.2021).
      3. Asp.net: официальная документация [Электронный ресурс] URL: [https://dotnet.microsoft.com/learn/aspnet](https://dotnet.microsoft.com/learn/aspnet%20) (дата обращения: 17.04.2021).
      4. Статья на habr.com [Электронный ресурс] URL: <https://habr.com/ru/post/469635/> (дата обращения: 17.04.2021).
      5. Федеральный закон о "О национальной платежной системе" [Электронный ресурс] URL: <https://rg.ru/2011/06/30/fz-dok.html> (дата обращения: 19.04.2021).
      6. Видео по теме на youtube.com [Электронный ресурс] URL: <https://www.youtube.com/watch?v=Uco05GknIUU> (дата обращения: 03.06.2021).

# Приложение А

Листинг программы

## CardTran.cshtml

@model \_012.Models.Card

@{

ViewData["Title"] = "Edit";

}

<h1>Edit</h1>

<h4>Card</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form **asp-action**="Edit">

<div **asp-validation-summary**="ModelOnly" class="text-danger"></div>

<input **type**="hidden" **asp-for**="Id" />

<div class="form-group">

<label **asp-for**="CardNumber" class="control-label"></label>

<input **asp-for**="CardNumber" class="form-control" />

<span **asp-validation-for**="CardNumber" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="CardAccount" class="control-label"></label>

<input **asp-for**="CardAccount" class="form-control" />

<span **asp-validation-for**="CardAccount" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="CardBalance" class="control-label"></label>

<input **asp-for**="CardBalance" class="form-control" />

<span **asp-validation-for**="CardBalance" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="CardHolderId" class="control-label"></label>

<input **asp-for**="CardHolderId" class="form-control" />

<span **asp-validation-for**="CardHolderId" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="CardHolderName" class="control-label"></label>

<input **asp-for**="CardHolderName" class="form-control" />

<span **asp-validation-for**="CardHolderName" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="expiryDate" class="control-label"></label>

<input **asp-for**="expiryDate" class="form-control" />

<span **asp-validation-for**="expiryDate" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="cvi" class="control-label"></label>

<input **asp-for**="cvi" class="form-control" />

<span **asp-validation-for**="cvi" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="changeBalance" class="control-label"></label>

<input **asp-for**="changeBalance" class="form-control" />

<span **asp-validation-for**="changeBalance" class="text-danger"></span>

</div>

<div class="form-group">

<input type="submit" value="Save" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div>

<a **asp-action**="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}