# Введение

Учет личных финансов позволяет внести элементы организации в собственные денежные траты и доходы. Любой может учитывать движение собственных денежных средств не прибегая, ни к каким информационным технологиям. Однако, отслеживание трат с помощью электронных гаджетов гораздо быстрее и удобнее. В данном случае мы можем воспользоваться собственным телефоном, планшетом или компьютером. На любом из этих устройств можно создать приложение, которое будет вести доходы и расходы. Но можно создать приложение, которое будет работать независимо от устройства и операционной системы. В данном случае речь идет о создании клиент серверного приложения, которое может поддерживать работу с множеством клиентов. Создание информационной системы это комплексный процесс, который разделяется на подзадачи, каждая из которых подобно кирпичику встраивается в работу. В данном проекте список подзадач будет выглядеть следующим образом:

* работа с предметной областью;
* нужно будет определиться, что именно из реальной жизни и каким образом перенесется в программную среду;

Кроме теоретической базы будут рассмотрены и технические вопросы, которые решаются при разработке ПО.

* Необходимо будет определится какая информация должна сохраняться и способы сохранения.
* механизмы, которые определяют способы взаимодействия с пользователей с системой.

Цель данного проекта – это разработка информационной системы. Объект исследования – сам процесс разработки, те шаги, которые должно пройти приложение, от идеи до реализации. Для того чтобы достигнуть цели, необходимо проанализировать требования к приложению. После анализа требований, на каждую выявленную проблему нужно рассмотреть список возможных вариантов ее решения. И далее выбрать наиболее подходящий план действий. Поэтому в ходе работы над проектом будут рассмотрены следующие вопросы:

* анализ требований к системе;
* разновидности СУБД и выбор наиболее подходящей для данного проекта;
* анализ технологий для реализации информационной системы;
* выбор наиболее подходящего технического решения для данной информационной системы;
* взаимодействие приложения и СУБД;

После того как выбор необходимых технологий и требований будет завершен, последует проектирование информационной системы. Где обозначатся взаимодействующие программные сущности. Данные объекты лягут в основу базы данных в виде таблиц и выстроенных отношений. Кроме этого будет необходимо выстроить архитектуру программного комплекса.

После завершения проектирования на всех уровнях, можно будет приступать к реализации проекта. Таким образом, в данной работе будут рассмотрены основные шаги, которые проходят приложения от идеи до реализации.

# 1.Теоретическая часть

Учет личных финансов заключается в создании различных записей, которые относятся к доходам и расходам. Разумеется, доходы могут быть из разных источников, так же как и расходы имеют различные категории. Кроме доходов и расходов, можно иметь наглядный результат в виде отчета или общего баланса средств, который мог бы наглядно демонстрировать динамику трат и доходов в течении определенного периода.

## 1.1 Выработка требований к приложению

Чтобы не возникло проблем в реализации необходимо заострить внимание на подробных требованиях. Мы разделим требования к приложению по разным типам:

Функциональные:

* Так как у каждого пользователя будет свое рабочее пространство, то необходимо внедрить возможность регистрации и авторизации.
* Необходимо разграничить полномочия пользователей, потому что в приложении будет производится работа администратора. Поэтому необходимо ввести разграничение по ролям.
* Каждый пользователь может иметь возможность работы над своими записям
* Нужен механизм сохранения результатов работы пользователя, это может быть все что угодно. Прямое сохранение файлов, подключение к СУБД, подключение к SQLite.

Структурные

* кроссплатформенность – данная возможность позволит разворачивать проект серверного приложения без привязки к операционной системе;
* модульность – проект должен состоять из нескольких независимых или слабосвязанных частей, которые позволят работать с проектом в разных направлениях, исключив возможность повредить работоспособность всей системы;
* организация отдельного слоя для работы с данными – позволит изолировать часть проекта, где определена логика работы с данными;
* для хранения данных необходимо использовать бесплатные инструменты;
* проект может и должен разрабатываться без привязки к ОС – данное условие позволит оградить разработку от проблем, связанных с изменениями внешних условий;
* обработка ошибок должна быть исполнена на нескольких уровнях:

1. Клиентском – предупреждение при некорректно заполненной форме и блокировка отправки.
2. Серверном – проверка данных перед получением и отправкой, а также переадресация на случай нештатных ситуаций, критичные места обрамляются блоками try/catch.

## 1.2 Работа с данными

При создании текущей базы данных, еще на этапе концептуального проектирования был задел сразу на третью нормальную форму. Чтобы все таблицы не дублировали данные друг друга, а имели лишь ссылки. Единственное допущение, которое было при определении сущностей «клиент» и «сотрудник» - это единственное поле для ФИО, чтобы не перегружать информацией графическую демонстрацию отношений сущность – связь. В остальном все необходимые поля в таблицах не делимы.

Для хранения и управления информацией с проектируемой информационной системой можно выделить несколько объектов:

* категории доходов
* записи о доходах
* отчеты,
* роли пользователей,
* пользователи,
* категории расходов
* записями расходов.

Данные объекты обладают различными характеристиками. Это могут быть скалярные свойства, такие как дата и сумма прихода/расхода или название или описание. Также сущности могут ссылаться друг на друга, образуя различные связи или отношения. Прежде чем приступать к проектированию, рассмотрим несколько важных понятий для работы с базами данных.

### Отношение один ко многим

Не смотря на простоту данного понятия при проектировании базы данных возникает вопрос, о том какая из таблиц должна хранить информацию о другой таблице. Чтобы внести ясность в подход можем рассмотреть две таблицы «пользователь» и «запись дохода». С одной стороны пользователь должен знать, что он купил, поэтому в записи клиента должна быть ссылка на таблицу с доходами . С другой стороны, запись дохода также должна содержать данные о том пользователе, что ее создал. Чтобы определиться с этим вопросом, мы можем вспомнить советский мультфильм об «осьминожках». Суть в том, что папы «осьминожек» перепутали своих детей, так как они были многочисленны, имели одинаковую форму и меняли цвет. Из затруднительной ситуации помогли найти выход подоспевшие мамы, к которым устремились дети. В результате чего обе семьи оказались в сборе. Данная ситуация учит тому, что в отношении «один ко многим» каждая порожденная сущность должна хранить в себе информацию о своем родителе. И наоборот, родительская запись не должна иметь информацию о записях в дочерних таблицах, если в этом нет необходимости. Таким образом, запись пользователя всегда будет одна, но записей о доходах или расходах может быть множество, и каждая из них должна хранить ссылку на пользователя, то же условие соблюдается при отношении «категория доходов» - «доход» или «роль» - «пользователь» и т.д.

### Ключи

Первичные и внешние ключи - важные компоненты реляционной базы данных, используемые для установления связей между различными таблицами. Первичный ключ - это столбец или набор столбцов в таблице, который однозначно идентифицирует каждую запись в таблице. Он обеспечивает целостность данных и служит точкой отсчета для других таблиц. Первичный ключ всегда уникален и не может содержать нулевых значений. Данное поле должно гарантировать уникальность записи не смотря на сходства других полей. А это значит, что при определении первичного ключа необходимо запустить такой механизм, гарантирующий уникальность ключа. В большинстве СУБД реализована возможность автоматического увеличения ключа. Количество увеличения значения может регулироваться, однако в большинстве случаев, прирост составляет единицу. Также стоит отметить что при удалении записи значения первичнфых ключей остаются неизменными. В приведенном сценарии таблицы в качестве первичного ключа определен столбец "id".

Внешний ключ - это столбец или набор столбцов в таблице, который ссылается на первичный ключ другой таблицы. Он устанавливает связь между двумя таблицами, обеспечивая согласованность и целостность данных. Внешний ключ гарантирует, что значения в ссылающемся столбце совпадают со значениями в ссылающемся столбце первичного ключа. В рассматриваемых таблицах столбцы "id\_client", "id\_in\_come\_cat" и "id\_role" определены как внешние ключи, ссылающиеся на первичные ключи в других таблицах.

Внешние ключи обычно помогают создавать связи между таблицами, ссылаясь на первичный ключ другой таблицы. Они используются для обеспечения ссылочной целостности и поддержания согласованности данных в связанных таблицах.

### Нормализация

Нормализация базы данных - это процесс организации данных для обеспечения их целостности, устранения избыточности и повышения эффективности. Она предполагает разбиение базы данных на множество таблиц и определение связей между ними на основе специальных правил и форм нормализации. Наиболее часто используемыми формами нормализации являются:

Первая нормальная форма (1НФ):

- Исключить дублирование строк.

- Убедиться, что каждый столбец содержит атомарные значения

- Определить первичный ключ для каждой таблицы.

2. Вторая нормальная форма (2НФ):

- Выполнить все требования 1НФ.

- Убедиться, что все столбцы, не являющиеся ключами, функционально зависят от всего первичного ключа, а не только от его части.

3. Третья нормальная форма (3НФ):

- Удовлетворяет всем правилам 2НФ.

- Устранить транзитивные зависимости; обеспечить, чтобы столбцы без ключей зависели только от первичного ключа, а не от других столбцов без ключей.

Нормализация базы данных позволяет свести к минимуму избыточность данных, что приводит к улучшению их согласованности и целостности. Кроме того, повышается гибкость и производительность запросов и обновления данных. Однако важно найти баланс между нормализацией и денормализацией, исходя из конкретных требований приложения и базы данных.

### Создание таблиц

Мы уже познакомились с основными объектами, теперь можно на их основе выделить ключевые характеристики и перенести их в таблицы

1. В таблице для категории доходов хранит в себе название для источников дохода. Она первичный ключ для уникального идентификатора и столбец для хранения названия категории.
2. В таблице доходов хранятся следующие записи: первичный ключ, сумма дохода, дата дохода, ссылка для связи дохода с конкретным клиентом и внешний ключ для связи дохода с категорией.
3. Таблица отчетов используется для хранения отчетов, связанных с доходами и расходами. Она содержит первичный и внешний ключ для привязки отчета к конкретному клиенту, а также текстовые поля, которые могут быть задействованы для работы с доходами и расходами за определенный период, а также текстовое поле для хранения дополнительных сведений об отчете.
4. Имеется таблица где хранятся роли пользователей. Она имеет первичный ключ и столбец для хранения имени роли.
5. В таблице для пользователей хранится информация о пользователях, включая учетные данные для входа в систему. Она включает такие столбцы, как первичный ключ, поля для логина и пароля и ссылка для привязки пользователя к определенной роли.
6. Также имеется таблица, где хранятся категории расходов, она является противоположностью для таблицы с доходами, также как и категории расходов.

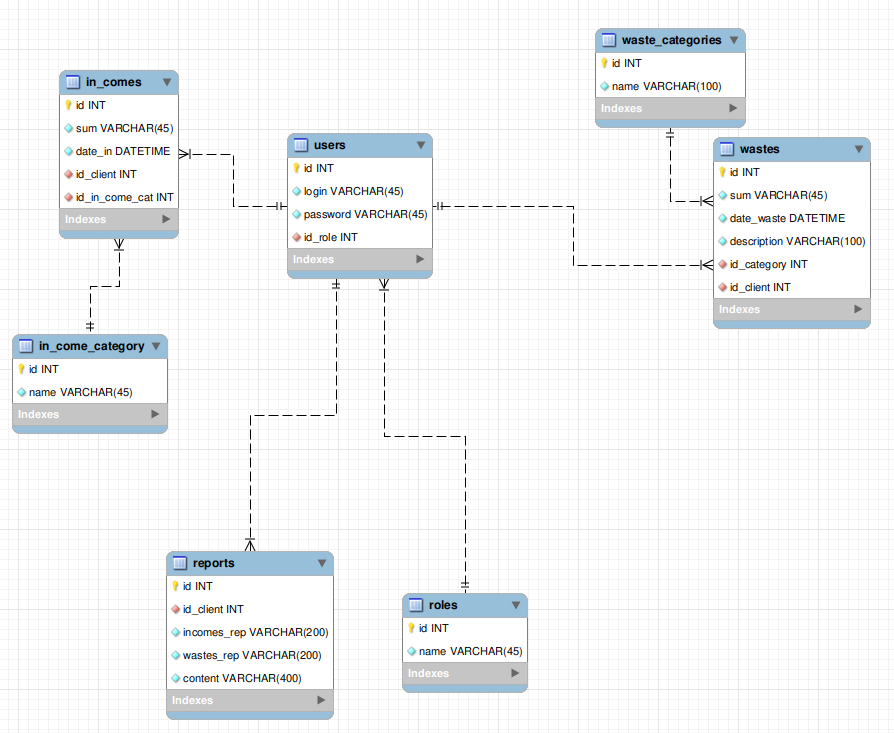


Рис 1. Диаграмма спроектированных таблиц и их связей для учета финансов.

## 1.3 Проектирование информационной системы

### Клиент серверное взаимодействие

Информационная система представляет из себя клиент серверное приложение. Пользователи системы взаимодействуют с ней с помощью отправки различных URL запросов. В приложении реализованы механизмы для обработки запроса в зависимости от его содержания и формы. Также в приложении имеются средства для работы с сервером базы данных. Чтобы выполнить запрос нужно ввести в строку браузера адрес запрашиваемой страницы. Браузер данную строку преобразует в запрос в виде:

[тип запроса, например GET][url адрес, например https:/hellow.html][версия протокола : HTTP/1.2]

Кроме этого формируется дополнительная информация о браузере, операционной системе и IP компьютера, с которого отправлен запрос. Из IP можно узнать локацию устройства. Таким образом, некоторые ресурсы блокируют пользователей из определенных стран. После того как сервер обработает запрос, он формирует ответ в котором содержится код ответа, дата и содержание ответа. Это может быть либо html код, который браузер преобразует в страницу, либо данные в различных форматах JSON или XML.

Интернет ресурсы по структуре могут быть устроены по-разному. Например, есть множество сайтов, состоящих из одной страницы, на которой отображается контактная информация без какой либо встроенной логики- афиши, объявления, сайты визитки и т.д. Есть также сайты с одной отображаемой страницей, но в данном случае на странице может быть задействовано большое количество технологий:

* формы для отправки данных на сервер;
* блоки с отображаемой информацией, которая загружается в реальном времени;
* механизмы, которые позволяют отправлять и загружать данные без перезагрузки страницы;

Также распространены сайты с множеством отображаемых страниц. В нашем случае информационная система не содержит множество различных статических web-страниц, а предоставляет лишь несколько ответов, которые могут содержать сгенерированную сервером информацию в соответствии запросу пользователя.

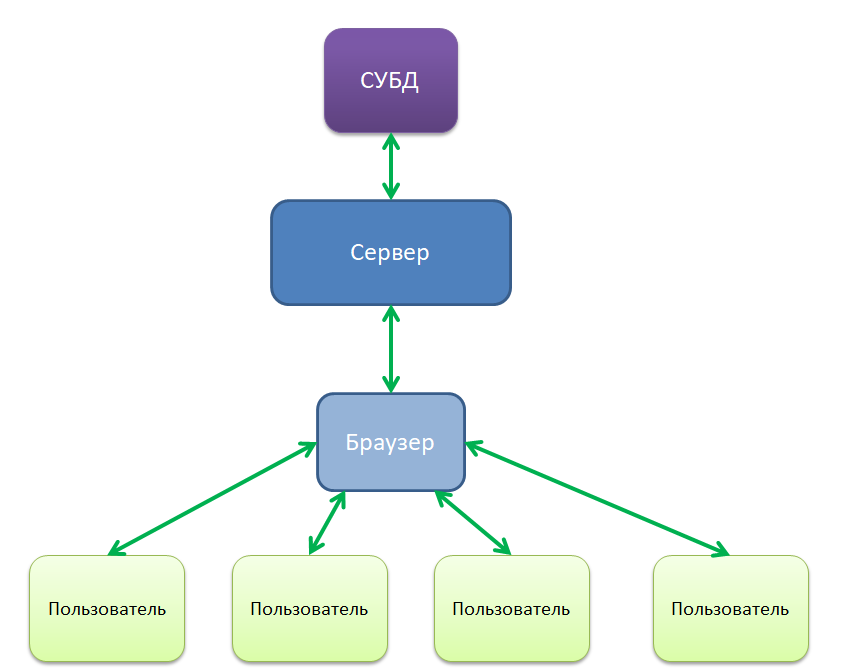


Рис 2. Схема работы информационной системы

Данная схема работы ИС позволяет взаимодействовать большому числу пользователей одновременно без ограничений операционной системы и типа устройства. Подобный ход увеличивает пользовательскую аудиторию, а значит, расширяет клиентскую базу приложения.

### Выбор и обоснование архитектурного паттерна проектирования ИС

Большинство клиент серверных приложений, в которых присутствует бизнес логика, спроектированы на базе самого распространенного паттерна MVC(Модель – Вид - Контроллер). Данный подход проверен временем и доказал свою эффективность. Ведь одно из самых главных и необходимых качеств информационной системы – это возможность к изменению и расширению. MVC состоит из нескольких типов программных механизмов, которые позволяют работать в системе в нескольких направлениях одновременно, без опасения разрушить всю систему разом. Так как архитектура MVC предполагает построение приложение не в виде крупных блоков, связанных между собой, а работу с каждой задачей отдельно. На каждый пользовательский запрос у нас будет отдельный метод контроллера и отдельное представление. Работа с базой данных осуществляется через классы моделей данных, которые хранят в себе те же характеристики, что и таблицы в СУБД. Поэтому создание, отладка и тестирование приложения проходит проще, так как мы не влияем на остальные механизмы без необходимости. Поэтому во многих web фреймворках в реализации используют именно этот паттерн в системе соглашений. Также часть сред разработки автоматически формируют соответствующий каркас при выборе соответствующего типа приложения.

# 2.Практическая часть

### 2.1 Программная реализация моделей данных приложения и сопоставление с таблицами в бд.

Так как база данных спроектирована, то мы можем согласно с таблицами в ней создать одноименные классы моделей вручную. Но есть и другой способ создания моделей данных из уже имеющейся базы. Entity Framework обладает инструментами, которые автоматически генерируют модели и контекст данных из указанной строки соединения. Для выполнения команд в среде разработки предусмотрена консоль менеджера пакетов. Если ввести в нее следующую строку:

Scaffold-DbContext "срока соединения с базой данных" Pamelo.EntityFrameworkCore.MySql -OutputDir Models

то при отсутствии ошибок в проекте приложения появятся все необходимые модели данных:

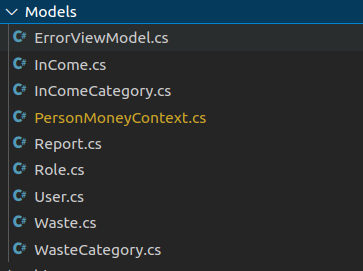


Рис 3. Список сгенерированных моделей в проекте.

Подобный инструмент генерации кода может существенно упростить разработку приложения, особенно если мы создаем проект от существующей базы данных. Но стоит отметить, что полагаться на подобные средства стоит ли при автоматизации простых рутинных задач. В таком случае мы рискуем ”захламить” проект моделями, контроллерами и представлениями, которые не будут использоваться или будут, но совсем иначе.

### 2.2 Реализация контроллеров

Работа контроллеров связана с передачей информации для определенной модели. Данные классы содержат методы, которые направлены на обработку пользовательских запросов.

Как правило, методы согласуют действия не только с запросами от пользователя, но и формируют работу с базой данных. Рассмотрим контроллер для работы с моделью данных “Пользователи” - UsersController.

Данный класс, как и большинство подобных содержит методы для добавления, просмотра, редактирования и удаления записей. Мы уже упоминали, что для работы с базой данных в контроллер помещается объект контекста:

public class UsersController : Controller

{

private readonly PersonMoneyContext \_context;

public UsersController(PersonMoneyContext context)

{

\_context = context;

}

Для того чтобы отобразить список пользователей в базе данных используется метод Index. В нем нет ничего сложного, нам достаточно обратится к контексту, получить от него список. И отправить данную информацию в представление, которое будет возвращено пользователю.

public async Task<IActionResult> Index()

{

var personMoneyContext = \_context.Users.Include(u => u.IdRoleNavigation);

return View(await personMoneyContext.ToListAsync());

}

Кроме пользователей из базы данных подтягиваются и соответствующие роли. Чтобы отобразить одноименное представление, тип возвращаемого значения в методы должен быть либо ActionResult, либо Task<IActionResult> при вызове асинхронного метода. Далее для работы с поступающими данными от контроллера в представлении необходимо указать модель данных, к которой мы будем обращаться:

@model IEnumerable<person\_money.Models.Product> - Представление получает перечисление типа “Product”.

Далее мы создаем таблицу:

@model IEnumerable<person\_money.Models.User>

<p>

<a class="btn btn-success" asp-action="Create">Создать</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Login)

</th>

<th>

@Html.DisplayNameFor(model => model.Password)

</th>

<th>

@Html.DisplayNameFor(model => model.IdRoleNavigation)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Login)

</td>

<td>

@Html.DisplayFor(modelItem => item.Password)

</td>

<td>

@Html.DisplayFor(modelItem => item.IdRoleNavigation.Id)

</td>

<td>

<a class="btn btn-dark" asp-action="Edit" asp-route-id="@item.Id">Редактировать</a> |

<a class="btn btn-info" asp-action="Details" asp-route-id="@item.Id">Подробнее</a> | |

<a class="btn btn-danger" asp-action="Delete" asp-route-id="@item.Id">Удалить</a>

</td>

</tr>

}

</tbody>

</table>

Для заголовка таблицы будут использоваться названия свойств типа, который предоставляет контроллер. Далее с помощью движка для работы с представлениями Razor, в цикле мы обратимся к получаемому списку и перебираем все его элементы прямо в коде представления.

Стоит отметить, что не все свойства данного типа скалярные, поэтому часть номеров внешних ключей отображается не в текстовых полях а в теге <select> для ограничения выбора при создании записи. Данные методы работы очень облегчают создание пользовательских интерфейсов, так как грань между серверным кодом и дизайнерскими решениями может быть практически стерта. В то же время можно сделать отдельный блок кода с данными, без привязки к отображаемым стилям.

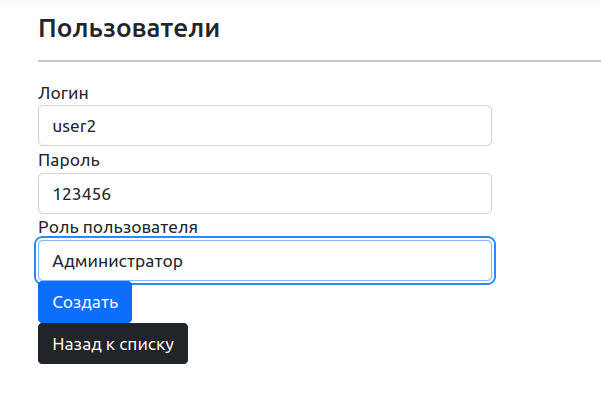


Рис 4. Добавление записи администратора

Кроме отображения данных контроллер предоставляет методы для внесения и редактирования записей. При успешном вводе данных мы можем видеть результат работы данных методов:

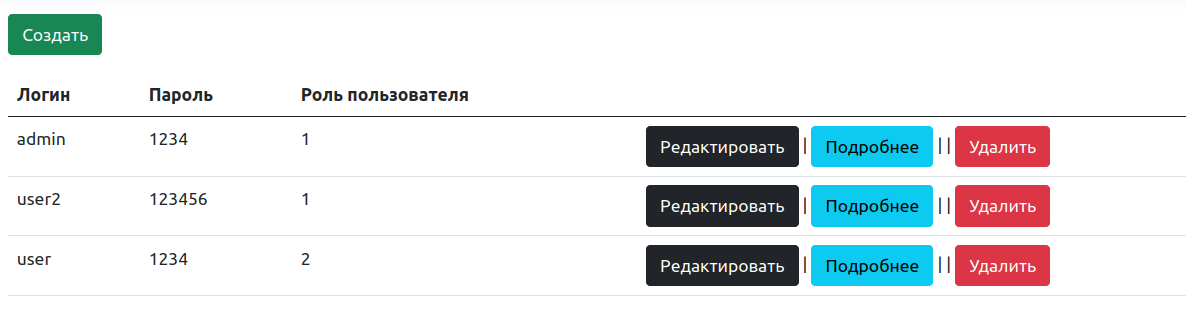
\

Рис 5. Содержимое списка пользователей

### 2.3 Краткое описание работы приложения

После реализации контроллеров и представлений. Можно подвести определенные итоги. Было разработано клиент серверное приложение с использованием паттерна MVC на языке C# и серверной платформой .NET 7.

Приложение работает следующим образом. Браузер отправляет URL запрос клиента при обращении к стартовой странице. Данный запрос обрабатывается в специальном классе – контроллере, который в качестве ответа предоставляет файл с расширением cshtml - обычная web страница, содержащая код C#, далее с помощью специального механизма по обработке данных файлов Razor, из кода C# появляется информация или новые элементы, которые в итоге и отправляются браузеру в обычном виде. После чего перед клиентом отображается html страница с набором ссылок и полей для ввода. Каждое из подобных действий запускает механизмы контроллеров, которые заново обрабатывают запрос и предоставляют иную информацию. Далее цикл повторяется.

Серверная сторона в свою очередь взаимодействует с базой данных и предоставляет контроллерам нужную информацию и предает элементам страниц необходимое содержание, а также добавляет обработку на различные события. Данные действия пользователя сохраняются в скрытых полях, а затем передаются в работу контроллерам, и далее сохраняются в базе данных. Так как используется механизм авторизации, то каждый из пользователей работает со своей частью системы.

# Заключение

В данной работе мы реализовали приложение для учета личных финансов. После реализации можно сделать несколько выводов.

Любое программное приложение, имеющее цель облегчить или автоматизировать часть работы из реальной жизни должно быть подкреплено знаниями предметной области, пониманием выбранной технологии для реализации приложения. Кроме того, перед кодированием мы должны провести аналогию между объектами в реальной жизни и программными сущностями. Произведено проектирование приложения и выбрана подходящая архитектура. Данная работа позволит оставить место для расширения возможностей приложения и добавит больше возможностей при параллельной разработке приложения.

В ходе данной работы были решены следующие задачи:

* Созданы модели, отражающие характеристики и поведение объектов из реальной жизни.
* Реализованы контроллеры – механизмы обеспечивающие работу с пользователями.
* Построены представления – файлы, которые формируются в качестве ответа пользователю.
* Разработана структура базы данных, в которой содержится вся необходимая информация.
* Реализовано разделение пользователей по ролям и авторизация.

Для реализации приложения задействованы следующие технологии:

* язык программирования C#;
* фреймворк для построения web приложений .NET Core;
* фреймворк для работы с СУБД Pomelo.EntityFrameworkCore.MySql;
* СУБД MySQL;
* библиотека каскадных таблиц стилей Bootstrap – для придания благоприятного внешнего вида представлениям и адаптивной верстки;
* язык программирования JavaScript и библиотека JQuery для работы с коррекцией ввода на стороне клиента.

Список задействованных технологий относится к открытому программному обеспечению и кроссплатформенности. Данный набор позволяет вести разработку и развертывать приложения на любой операционной системе. Санкции коснулись не только материальных ресурсов, но и различных информационных технологий. Поэтому команда, которая при разработке приложений пользуется технологиями, доступными по платной подписке, рискует остаться без инструментов и потерять проект со всеми наработками, так как используемые технологии могут быть заблокированы.

Чем дольше ведется работа над подобными приложениями, тем больше приходит понимание того, что хорошо спланированная архитектура и разделение большей части приложения на более мелкие и автономные блоки поможет избежать краха всей системы из за отказа одной из ее частей. Данный проект не идеален и может быть доработан. Хотелось бы в будущем расширить функционал и добавить интерактивности и отображения графиков, но базовый функционал уже есть и есть понимание технологий для дальнейшего развития данной платформы.