**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc121995510)

[1 Общая часть 4](#_Toc121995511)

[1.1 Анализ существующих решений по автоматизации предметной области 4](#_Toc121995512)

[1.2 Выбор методолгии разработки программного обеспечения 5](#_Toc121995513)

[1.3 Анализ предметной области 11](#_Toc121995514)

[1.4 Сбор требований 11](#_Toc121995515)

[1.4.1 Интервьюирование сотрудников предприятия 12](#_Toc121995516)

[1.4.2 Пользовательские истории 14](#_Toc121995517)

[1.5 Анализ и моделирование требований 16](#_Toc121995518)

[1.6 Спецификация требований 18](#_Toc121995519)

[1.7 Теоретическая информация о системах управления базами данных. 21](#_Toc121995520)

[1.7.1 Информация о существующих системах управления баз данных 24](#_Toc121995521)

[1.7.2 Сравнение обозреваемых систем управления базами данных 25](#_Toc121995522)

[2 Специальный раздел 28](#_Toc121995523)

[2.1 Архитектурное проектирование 28](#_Toc121995524)

[2.2 Проектирование базы данных 29](#_Toc121995525)

[2.3 Проектирование пользовательского интерфейса 32](#_Toc121995526)

[2.4 Проектирование модулей программного обеспечения 38](#_Toc121995527)

[2.5 Реализация программного обеспечения 40](#_Toc121995528)

[2.6 Реализация базы данных 43](#_Toc121995529)

[2.7 Тестирование приложения 45](#_Toc121995530)

[ЗАКЛЮЧЕНИЕ 50](#_Toc121995531)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 51](#_Toc121995532)

[ПРИЛОЖЕНИЕ А (обязательное) Листинг кода программы 52](#_Toc121995533)

[ПРИЛОЖЕНИЕ Б (обязательное) Листинг HTML-разметки 86](#_Toc121995534)

[ПРИЛОЖЕНИЕ В (обязательное) Листинг SQL-дампа базы данных 89](#_Toc121995535)

# ВВЕДЕНИЕ

В современных условиях хозяйствования, обусловленных нестабильной экономической средой, развитием новых форм предпринимательства, резко возрастают требования к качеству информационного обеспечения управления. Переход от существовавшей к новой системе информационного обеспечения предприятий привел к своеобразному информационному кризису, реальным выходом из которого является использование информационных систем и технологий.

В курсовом проекте необходимо произвести сбор данных о предметной области, выполнить проектирование программного продукта и разработать программное обеспечение.

На данном этапе тема курсового проекта является актуальной.

# Общая часть

## 1.1 Анализ существующих решений по автоматизации предметной области

На данный момент на рынке существуют множество средств для расчёта заработной платы на предприятии. Наиболее популярные из них являются «1С: Предприятие» и **«1С: Бухгалтерия».**

1С:Предприятие – программный продукт компании 1С, предназначенный для автоматизации деятельности на предприятии.

1С: Предприятие - это и технологическая платформа, и пользовательский режим работы. Технологическая платформа предоставляет объекты (данных и метаданных) и механизмы управления объектами. Объекты (данные и метаданные) описываются в виде конфигураций. При автоматизации какой-либо деятельности составляется своя конфигурация объектов, которая и представляет собой законченное прикладное решение. Конфигурация создаётся в специальном режиме работы программного продукта под названием «Конфигуратор», затем запускается режим работы под названием «1С: Предприятие», в котором пользователь получает доступ к основным функциям, реализованным в данном прикладном решении (конфигурации).

Достоинствами «1С: Предприятие» является:

* платформа приспособлена под российское законодательство и позволяет легко подстраиваться под регулярно меняющиеся законы;
* обладает высокой производительностью, что дает возможность решать с ее помощью самые сложные задачи;
* возможность использовать MS SQL Server.

Недостатки:

* достаточно сложна в освоении и требует специального обучения пользователей;
* затруднен поиск ошибок, сделанных во время обработки документов;
* программа является платной;
* из-за уникальности предприятий конфигурации требуют доработки.

**Программа «1С: Бухгалтерия» предназначена для комплексной автоматизации бухгалтерского, налогового, управленческого, кадрового, складского и оперативного учета на предприятии в полном соответствии с требованиями бухгалтерского, налогового и трудового законодательства.**

Достоинства программы:

* формирование отчётов в Microsoft Exel;
* высокая производительность.

Недостатки программы:

* программа является платной;
* программе не предусмотрено взаимодействие с удалённым сервером базы данных.

Рассматривая аналогичные прикладные решения можно сделать вывод, что они не удовлетворяют требованиям поставленной задачи. В связи с этим принято решение разработать программу «Расчёт зарплаты» в рамках данного курсового проекта, которая позволит вести учет заработной платы.

## 1.2 Выбор методолгии разработки программного обеспечения

В современном мире существует множество методологий разработки программного обеспечения. Среди всех выделяют семь основных:

* Waterfall Model – каскадная модель (модель – «водопад»). Одна из старшейших методологий разработки, подразумевающая последовательное прохождение стадий, каждая из которых должна завершаться до начала следующей. Благодаря её жесткости разработка проходит быстро, стоимость и срок заранее определены. Недостатками данной методологии является жесткость (результат достижим только в проектах с четко и заранее определенными требованиями и способами реализации без возможности возврата к предыдущей стадии разработки), наличие недочетов при выборе метода без основательного её выбора о которых становится известно лишь при завершении проекта. Стоимость изменений на этапе производства чрезвычайно высока, из-за невозможности их произвести до окончания всех процессов. Тем не менее, фиксированная стоимость часто преобладает над минусами методологии.
* V – унаследовала структуру «этап за этапом» от каскадной модели. Применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления подушками безопасности в транспортных средствах и так далее. Особенностью считается то, что она направлена на тщательную проверку и тестирование продукта, находящегося уже на первоначальных стадиях проектирования. Стадия тестирования проводится одновременно с соответствующей стадией разработки, например, во время реализации программного обеспечения проводится покрытие модульными тестами.
* Incremental Model – Инкрементная модель. В инкрементной модели полные требования к системе делятся на различные сборки. Терминология часто используется для описания поэтапной сборки ПО. Имеют место несколько циклов разработки, и вместе они составляют жизненный цикл «мульти-водопад». Цикл разделен на более мелкие легко создаваемые модули. Каждый модуль проходит через фазы определения требований, проектирования, кодирования, внедрения и тестирования. Процедура разработки по инкрементной модели предполагает выпуск на первом большом этапе продукта в базовой функциональности, а затем уже последовательное добавление новых функций, так называемых «инкрементов». Процесс продолжается до тех пор, пока не будет создана полная система. Инкрементные модели используются там, где отдельные запросы на изменение ясны, могут быть легко формализованы и реализованы.
* RAD (Rapid Application development Model) Model – модель быстрой разработки приложений. RAD является разновидностью инкрементной модели. В RAD-модели компоненты или функции разрабатываются несколькими высококвалифицированными командами параллельно, будто несколько мини-проектов. Временные рамки одного цикла жестко ограничены. Созданные модули затем интегрируются в один рабочий прототип. Синергия позволяет очень быстро предоставить клиенту для обозрения что-то рабочее с целью получения обратной связи и внесения изменений.
* Agile Model – модель гибкой разработки. В «гибкой» методологии разработки после каждой итерации заказчик может наблюдать результат и понимать, удовлетворяет он его или нет. Это одно из преимуществ гибкой модели. К ее недостаткам относят то, что из-за отсутствия конкретных формулировок результатов сложно оценить трудозатраты и стоимость, требуемые на разработку. Экстремальное программирование (XP) является одним из наиболее известных применений гибкой модели на практике. В основе такого типа — непродолжительные ежедневные встречи — «Scrum» и регулярно повторяющиеся собрания (раз в неделю, раз в две недели или раз в месяц), которые называются «Sprint». На ежедневных совещаниях участники команды обсуждают отчеты о проделанной работы с конца последнего «Scrum», список задач, которые сотрудник должен выполнить до следующего собрания, затруднения в процессе работы. Методология хорошо подходит для больших и нацеленных на длительный жизненный цикл проектов, постоянно адаптируемых к условиям рынка.
* Iterative Model – итеративная или итерационная модель. Итерационная модель жизненного цикла не требует для начала полной спецификации требований. Вместо этого, создание начинается с реализации части функционала, становящейся базой для определения дальнейших требований. Этот процесс повторяется. Версия может быть неидеальна, главное, чтобы она работала. Понимая конечную цель, мы стремимся к ней так, чтобы каждый шаг был результативен, а каждая версия — работоспособна.
* Spiral Model – спиральная модель. «Спиральная модель» похожа на инкрементную, но с акцентом на анализ рисков. Она хорошо работает для решения критически важных бизнес-задач, когда неудача несовместима с деятельностью компании, в условиях выпуска новых продуктовых линеек, при необходимости научных исследований и практической апробации.

Визуализация методологий разработки программного обеспечения показана на рисунках 1–7.

Изображение выглядит как текст, визитка, снимок экрана

Автоматически созданное описание

Рисунок 1 - Визуализация каскадной методологии разработки ПО   
(Waterfall Model)



Рисунок 2 - Визуализация V-образной методологии разработки ПО

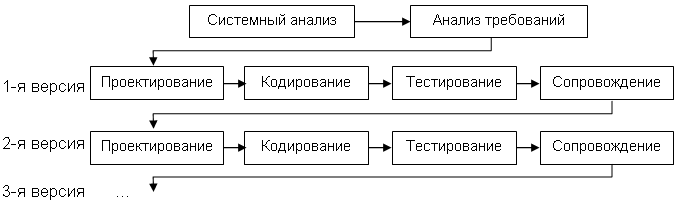


Рисунок 3 – Визуализация инкрементной модели разработки ПО

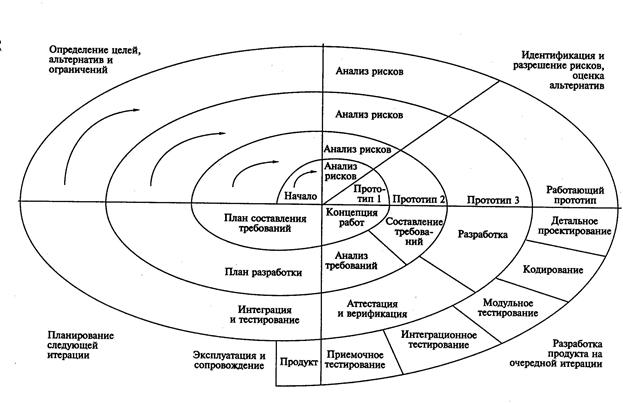


Рисунок 4 – Визуализация RAD – методологии разработки ПО

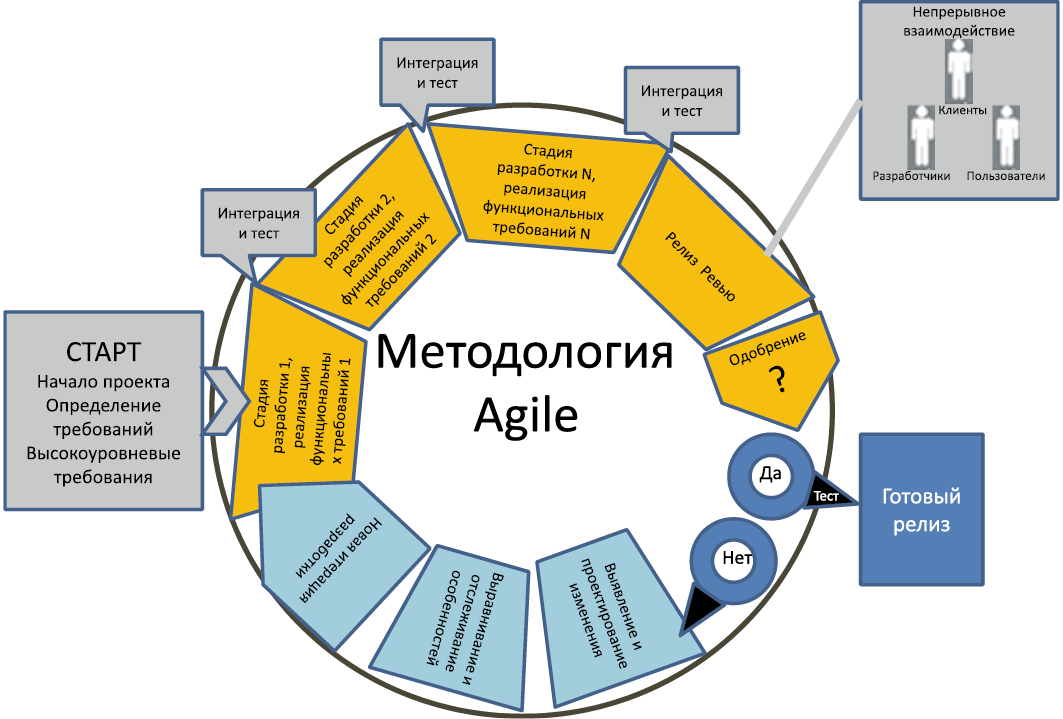


Рисунок 5 – Визуализация Agile – методологии разработки ПО

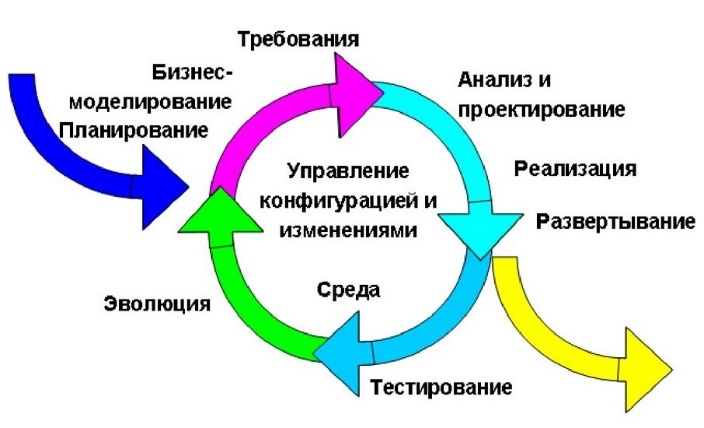


Рисунок 6 – Визуализация итеративной методологии разработки ПО

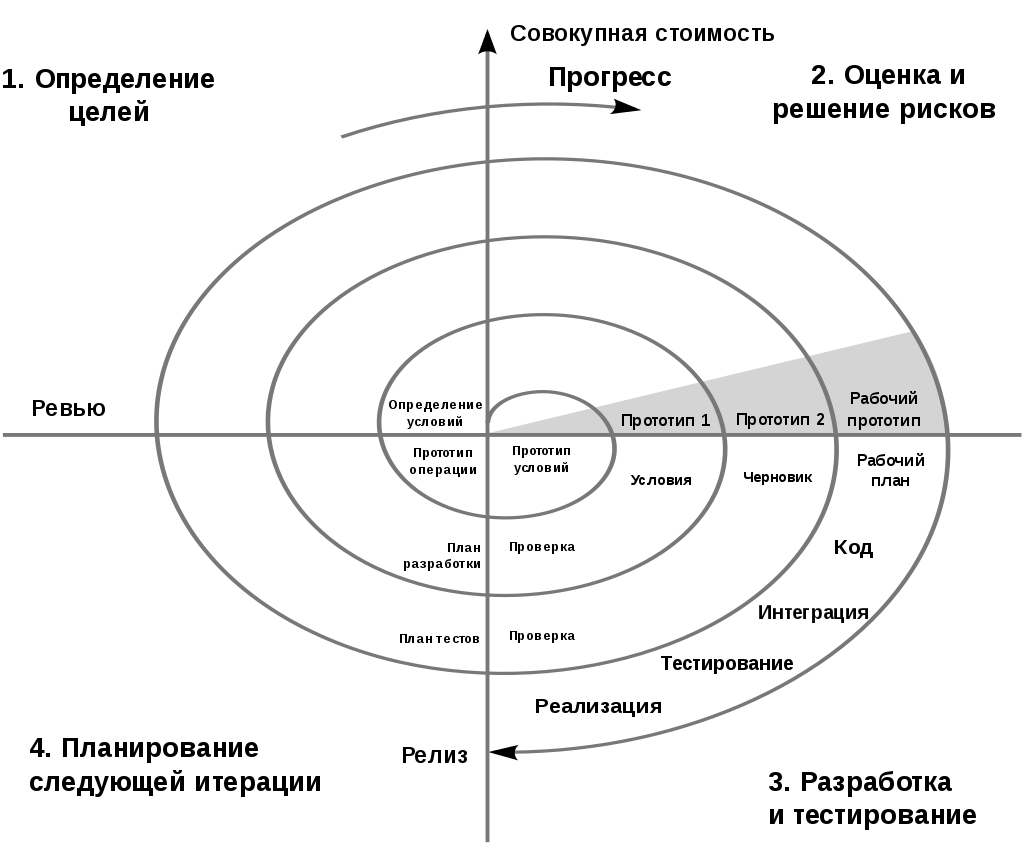


Рисунок 7 – Визуализация спиральной модели разработки ПО

## 1.3 Анализ предметной области

Программное обеспечение разрабатывается для нужд отдела бухгалтерии. Задачи, которые обычно решает бухгалтерия – начисление заработной платы, расчет налогового бремени, формирование отчетов.

Бухгалтерия — штатно-структурное подразделение хозяйствующего субъекта, предназначенное для аккумулирования данных о его имуществе и обязательствах. Бухгалтерия является источником документально обоснованной и структурированной экономической информации, необходимой для принятия управленческих решений в целях обеспечения эффективного хозяйствования.

Разрабатываемая АИС позволяет автоматизировать отдел Бухгалтерии в области расчета и начисления заработной платы, формирования отчетов.

При расчёте заработной платы используются тарифы, за определенный промежуток времени работы – оклад за месяц или почасовая ставка. Если для оплаты труда используется оклад за месяц, то сумма начисления зарплаты равна этому окладу; если же используется почасовая ставка, то чтобы рассчитать сумму начисленной зарплаты нужно умножить количество отработанных дней или часов на сумму тарифа оплаты труда за день или за час. Базовое количество часов работы сотрудника в день – 8. Сверхурочные (работа сверх базового количества часов) оплачивается с коэфициетом 1.5.

Сотрудникам из служащих с фиксированным окладом за месяц так-же выплачивается премия, исходя из обьема продаж. Процент вознаграждения определяется индивидуально для каждого служащего и может составлять 10%, 15%, 25% или 35%.

Задачи по удержанию налогового бремени не стоит.

## 1.4 Сбор требований

Чтобы успешно реализовать проект, необходимо корректно сформулировать требования к системе.

Требования — это описание необходимых или желаемых свойств продукта.

На этапе сбора требований основная работа ведется с заказчиком системы и её будущими пользователями. Цель этапа — точно определить функции продукта и способы его интеграции в существующие процессы.

Качественное выполнение работ на этом этапе гарантирует то, что будущий продукт будет соответствовать ожиданиям заказчика. Четкая расстановка приоритетов обеспечивает реализацию наиболее востребованной функциональности и исключение второстепенной/невостребованной функциональности, что сэкономит бюджет и сроки.

Сбор требований проводился методом интервьюирования, который заключается в беседе между разработчиком системы и заказчиком. Этот метод применяется, когда большим объемом знаний обладает ограниченный круг людей, и обычно используется для беседы с одним человеком.

Вместе с сбором требований был проведен анализ списка заинтересованных лиц и пользователей системы. Было установлено, что пользователями системы будут являться все сотрудники компании, заинтересованными лицами являются директор предприятия и все сотрудники компании.

Дополнительно был проведен сбор пользовательских историй.

### 1.4.1 Интервьюирование сотрудников предприятия

На таблицах 1–3 приведены вопросы и ответы из интервью со следующими сотрудниками предприятия – системным администратором, бухгалтером и рядовым сотрудником.

Сотрудник 1 – Старший системный администратор Иванов И. И., отдел технической поддержки пользователей.

Таблица 1 – Интервьюирование системного администратора

| **Вопросы** | **Ответы** |
| --- | --- |
| Какие аппаратно-программные комплексы, ПО и СУБД используются на предприятии? | У нас используются БД управления проектами, которая функционирует в среде DB2 на мейнфрейме IBM. |
| Требуется ли синхронизация ПО с действующей БД? | Требуется. |
| Какие данные содержатся в БД в среде DB2? | БД управления проектами содержит информацию относительно тарифов и проектов. |
| Какой уровень доступа должен быть у системного администратора | Полный доступ к системе. |
| Есть ли у вас предпочтения по СУБД для разрабатываемого ПО? | Да, было бы неплохо реализовать ПО с СУБД «Microsoft SQL Server», так-как наш персонал обучен к работе с ней. |

Сотрудник 2 – Старший бухгалтер Сидорова А. А., отдел бухгалтерии.

Таблица 2 – Интервьюирование бухгалтера

| **Вопросы** | **Ответы** |
| --- | --- |
| Какой требуется функционал для отдела бухгалтерии для обеспечения его работы? | В ПО должны быть реализованы отчеты по начислениям ЗП, по формирования отчетов об отработанном времени, просмотр банковских данных сотрудников. |
| Какие виды начислений заработной платы вы используете? | Почасовая и фиксированная ставка. |
| Как происходит назначение заработной платы? | Начисление заработной платы происходит на основе карточек учета отработанного времени сотрудников, которые содержат дату и количество отработанных часов.  Работникам с фиксированным окладом может быть назначена премия. Выплаты происходят на банковскую карту сотрудника. |

Сотрудник 3 – менеджер Петров П. П., отдел продаж.

Таблица 3 – Интервьюирование сотрудника

| **Вопросы** | **Ответы** |
| --- | --- |
| Какой требуется функционал вы хотели бы увидеть? | Я бы хотел изменять свои платежные данные без обращения в отдел бухгалтерии, менять способ расчета и самостоятельно формировать отчеты о выплатах. |
| Есть ли у вас специфические требования к интерфейсу? | Да, я иностранец, мне сложно изучать информацию на русском языке, было бы хорошо, если бы интерфейс дублировался и на английском языке. |

### 1.4.2 Пользовательские истории

Пользовательские истории (англ. User Story) — способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя.

Пользовательские истории используются гибкими методологиями разработки программного обеспечения для спецификации требований (вместе с приёмочными испытаниями). Каждая пользовательская история ограничена в размере и сложности. Часто история пишется на маленькой бумажной карточке. Это гарантирует, что она не станет слишком большой.

В Экстремальном программировании (методология XP) пользовательские истории пишутся пользователями (заказчиками) системы. В методологии SCRUM — проходят проверку пользователем в роли «Владелец продукта» (англ. Product Owner). Для заказчиков (пользователей) пользовательские истории являются основным инструментом влияния на разработку программного обеспечения.

Пользовательские истории — быстрый способ документировать требования клиента, без необходимости разрабатывать обширные формализованные документы и впоследствии тратить ресурсы на их поддержание. Цель пользовательских историй состоит в том, чтобы быть в состоянии оперативно и без накладных затрат реагировать на быстро изменяющиеся требования реального мира.

Далее приведены истории системного администратора, бухгалтера и сотрудника, принимавших участие в интервьюировании:

* Я как бухгалтер хочу систему отчетов для того, чтобы формировать общие отчеты по организации;
* Я как сотрудник хочу систему отчетов для того, чтобы сформировать отчет о своей заработной плате;
* Я как системный администратор хочу, чтобы в системе отчетов было удаление и редактирование для того, чтобы иметь возможность управления ими;
* Я как сотрудник хочу систему изменения платежных данных и их просмотр для того, чтобы отслеживать правильность ввода реквизитов и своевременно их редактировать;
* Я как системный администратор хочу систему изменение реквизитов для того, чтобы отслеживать правильность их ввода и своевременно их редактировать, и удалять;
* Я как бухгалтер хочу систему изменения реквизитов для того, чтобы легче формировать зарплатную ведомость и обеспечить максимальную корректность начисления;
* Я как системный администратор хочу систему личного кабинета для менеджмента сотрудников в системе;
* Я как сотрудник хочу систему личного кабинета для того, чтобы вносить изменения в журнал рабочего времени для повышения моего дохода, менять формат начисления зарплаты;
* Я как бухгалтер хочу систему электронный журнал рабочего времени для того, чтобы повысить корректность зарплат сотрудников и своевременного их премирования;
* Я как системный администратор хочу электронный журнал рабочего времени для того, чтобы повысить свою заработную плату и облегчить менеджмент данных;
* Я как сотрудник хочу выбор языка в программном обеспечении для того, чтобы я мог полностью воспринимать информацию;
* Я как системный администратор, хочу выбор языка в программном обеспечении для того, чтобы облегчить мою работу.

## 1.5 Анализ и моделирование требований

Для создания моделей используется UML — язык графического описания для объектного моделирования в области разработки программного обеспечения.

Цель анализа — качественно и подробно описать требования, которые позволят менеджерам реалистично оценить все затраты на проект, а разработчику — начать проектирование, разработку и тестирование

Диаграмма вариантов использования — диаграмма, на которой отражены отношения, существующие между актёрами и вариантами использования.

Основная задача — представлять собой единое средство, дающее возможность заказчику, конечному пользователю и разработчику совместно обсуждать функциональность и поведение системы. Диаграмма вариантов использования изображена на рисунке 8. Описание ролей изображено в таблице 4.

В информационной системе используются три роли:

* Администратор;
* Сотрудник;
* Бухгалтер.

Таблица 4 – Описание ролей в информационной системе

| **Название роли** | **Описание** |
| --- | --- |
| Администратор | CRUD всех частей программного обеспечения |
| Бухгалтер | CRUD отчеты по организации, CRUD назначение и изменение премирования, CRUD просмотр журнала рабочего времени |
| Сотрудник | Просмотр и изменение реквизитов, просмотр личного кабинета, просмотр отчета о выплатах заработной платы, внесение информации в журнал рабочего времени и его просмотр |

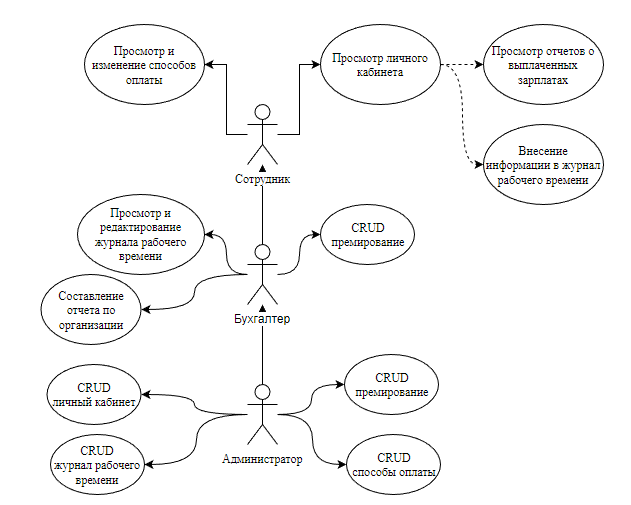


Рисунок 8 – UML-диаграмма распределения ролей

## 1.6 Спецификация требований

Важнейшей частью проектирования любого программного обеспечения является определение основных требований, предъявляемых к моделируемой системе.

Описание требований заказчика осуществляется по четырем категориям. Категории представлены и описаны в следующей таблице 5.

Таблица 5 – Категории описания требований

| **Категория** | **Описание** |
| --- | --- |
| F | Функциональные требования, описывающие требуемую функциональность или прецеденты системы |
| S | Системные требования, такие как используемые платформы |
| P | Требования к представлению |
| R | Требования, определяющие риски, которым должно быть уделено основное внимание при разработке системы |

Функциональные требования категории F представляют собой перечень сервисов, которые должна выполнять система. При этом необходимо указать, как система реагирует на входные данные.

Описание функциональных требований представлены в таблице 6.

Таблица 6 – Функциональные требования

|  |  |  |
| --- | --- | --- |
| **Требование** | **Тип** | **Описание** |
| Аутентификация пользователя | F | Для работы в системе необходимо пройти аутентификацию |
| Непротиворечивый ввод данных | F | Проверка типов данных на стадии ввода |
| Отчеты по требованию | F | Отчеты, которые запрашивают пользователи |

Второй категорией в описании требований является категория системных требований - S. Описания системных требований представлены в таблице 7.

Таблица 7 – Системные требования

| **Требование** | **Тип** | **Описание** |
| --- | --- | --- |
| Архитектура | S | Intel Core i3-12200 или более мощный |
| Платформа | S | Windows 10 |
| СУБД | S | MS SQL Server 2019 |
| Язык программирования | S | ASP .NET Framework  C# |
| Информационно-логический язык | S | Язык структурированных запросов SQL  Transact-SQL расширение языка SQL |

Требования к представлению (Р) относятся к третьей категории. Они описывают формирование требований заказчика к интерфейсу программного обеспечения. Описания требований к представлению показаны в таблице 8.

Таблица 8 – Требования к представлению

| **Требование** | **Тип** | **Описание.** |
| --- | --- | --- |
| Интерфейс рабочего окна | P | Мультиязычный интерфейс приложения на английском и русском языке |
| Корректный ввод данных | P | Данные несоответствующих типов не принимаются, выдается предупреждение |

К четвертой категории относятся требования к рискам (R). Данная категория требований направлена на общую безопасность системы и решает такие вопросы как сохранение непротиворечивости состояния баз данных, защиту от вторжений, резервное копирование и восстановление. Описания требований к рискам представлены в таблице 9.

Таблица 9 – Требования к рискам

| **Требование** | **Тип** | **Описание** |
| --- | --- | --- |
| Соответствие значений в таблицах внесенным данным | R | Поля в таблицах должны соответствовать типу введенных данных |
| Построение отчетов | R | Полное соответствие содержимому в таблицах |
| Сохранность и целостность данных | R | Система должна обеспечивать сохранность данных в случае непредвиденных сбоях |

На основе описания требований составляется документ, именуемый Спецификацией. Спецификация требований – документ, в котором точно указываются функции и возможности, которыми должно обладать ПО, а также необходимые ограничения.

Спецификация требований (Software Requirements Specification, SRS) используется для текущего сопровождения проекта и представления требований, сформулированных по отношению к проекту. SRS позволяет определить предметную область программного продукта, рассматриваемого относительно трех его основных составляющих: данных, процесса и поведения. Спецификация SRS позволяет от определения предметной области проекта перейти к области решений, определив три модели требований, отображающие характеристики данных, процесса и поведения. Данный документ должен содержать состав и наименование комплексов задач, требования по изменению организационной структуры, состав обеспечивающих подсистем.

## 1.7 Теоретическая информация о системах управления базами данных.

Для разработки баз данных требуется выбрать систему управления базами данных.

Система управления базами данных (СУБД) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Современная СУБД содержит в своем составе программные средства создания баз данных, средства работы с данными и сервисные средства. С помощью средств создания БД проектировщик, используя язык описания данных (ЯОД), переводит логическую модель БД в физическую структуру, а на языке манипуляции данными (ЯМД) разрабатывает программы, реализующие основные операции с данными (в реляционных БД – это реляционные операции). При проектировании привлекаются визуальные средства, т.е. объекты, и программа-отладчик, с помощью которой соединяются и тестируются отдельные блоки разработанной программы управления конкретной БД.

Средства работы с данными предназначены для пользователя БД. Они позволяют установить удобный (как правило, графически многооконный) интерфейс с пользователем, создать необходимую функциональную конфигурацию экранного представления выводимой и вводимой информации (цвет, размер и количество окон, пиктограммы пользователя и т.д.), производить операции с данными БД, манипулируя текстовыми и графическими экранными объектами. Состав СУБД отображен на рисунке 10.

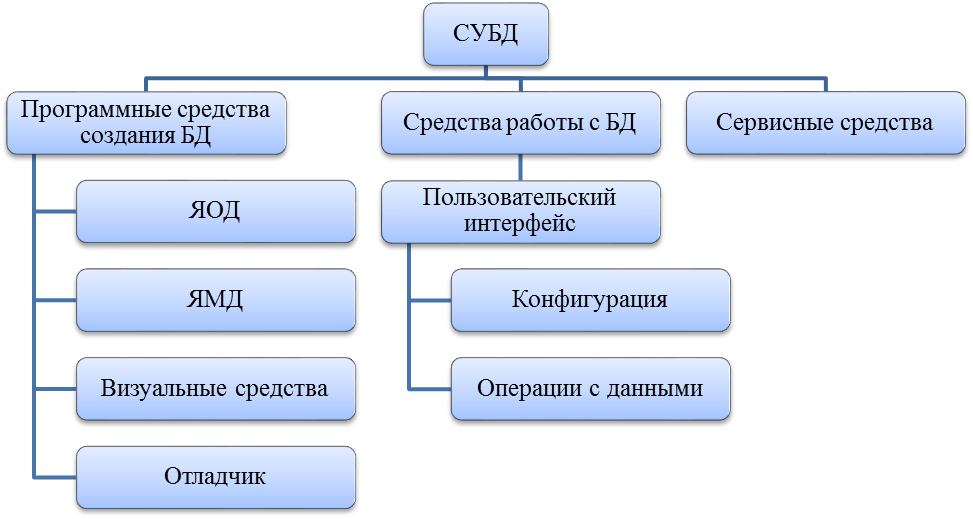


Рисунок 10 - Состав СУБД

В среде СУБД можно выделить следующих пять основных компонентов:

* Аппаратное обеспечение. Для работы СУБД и приложений необходимо некоторое аппаратное обеспечение. Одни СУБД предназначены для работы только с конкретными типами операционных систем или оборудования, другие могут работать с широким кругом аппаратного обеспечения и различными операционными системами. Для работы СУБД обычно требуется некоторый минимум оперативной и дисковой памяти, но такой минимальной конфигурации может оказаться совершенно недостаточно для достижения приемлемой производительности системы.
* Программное обеспечение. Этот компонент включает операционную систему, программное обеспечение самой СУБД, прикладные программы, включая и сетевое программное обеспечение, если СУБД используется в сети.
* Данные – наиболее важный компонент с точки зрения конечных пользователей. База данных содержит как рабочие данные, так и метаданные, т.е. "данные о данных".

Процедуры, к которым относят инструкции и правила, которые должны учитываться при проектировании и использовании базы данных: регистрация в СУБД; использование отдельного инструмента СУБД или приложения; запуск и останов СУБД; создание резервных копий СУБД; обработка сбоев аппаратного и программного обеспечения, включая процедуры идентификации вышедшего из строя компонента, исправления отказавшего компонента (например, посредством вызова специалиста по ремонту аппаратного обеспечения), а также восстановления базы данных после устранения неисправности; изменение структуры таблицы, реорганизация базы данных, размещенной на нескольких дисках, способы улучшения производительности и методы архивирования данных на вторичных устройствах хранения.

Пользователи: клиенты БД, администратор БД, прикладные программисты.

### 1.7.1 Информация о существующих системах управления баз данных

**1.7.1.1 Microsoft Access**

СУБД Microsoft Access – идеальная по простоте и удобству среда разработки баз данных. В качестве недостатков можно отметить отсутствие возможности формирование исполнимого файла, медленную работу с базами данных, скромные технические характеристики. В целом, СУБД Microsoft Access идеальна для обучения основам баз данных, но слабо применима в реальных проектах.

**1.7.1.2. Microsoft SQL Server**

СУБД Microsoft SQL Server – полнофункциональная серверная СУБД с прекрасными средствами защиты, администрирования, архивирования и восстановления баз данных, с языком запросов Transact-SQL. Обеспечивает надежное и компактное хранение, а так же высокопроизводительную обработку баз данных больших объёмов. Microsoft SQL Server поддерживает тиражирование данных, параллельную обработку, отличается простотой управления и использования. Имеются и некоторые технические недостатки, такие как отсутствие кроссплатформенности, ограничения на использование ЦП, ОЗУ, а так же ограничения на размер базы данных.

**1.7.1.3. PostgreSQL**

PostgreSQL включает в себя все плюсы SQL Server, но по сравнению с ней имеет преимущества в технических характеристиках, в частности поддерживает кроссплатформенность, не имеет ограничений на использование ЦП и ОЗУ, а так же ограничений на размер базы данных. Кроме этого, PostgreSQL – свободно распространяемая СУБД с открытым исходным кодом, в отличии от остальных представленных.

### 1.7.2 Сравнение обозреваемых систем управления базами данных

Имея теоретическую информацию, можно произвести выбор из существующих СУБД для разработки информационной системы. Результаты сравнения находятся в Таблице 10.

Сокращения, используемые в Таблице 10:

* СУБД – Система управления базами данных;
* ОС – Операционная система;
* ОЗУ – Оперативное запоминающее устройство;
* Гб – Гигабайт;
* Пб – Петабайт;
* БД – База данных;
* ЦП – Центральный процессор.

Таблица 10 – Сравнение обозреваемых СУБД

| **Наименование СУБД** | **Microsoft Access** | **Microsoft SQL Server Standart** | **PostgreSQL** |
| --- | --- | --- | --- |
| Кроссплатформенность (ОС Windows/Linux) | Поддержка только  ОС Windows | Поддержка только  ОС Windows | Имеется поддержка ОС  Linux и Windows |
| Ограничение на использование ОЗУ, Гб | отсутствует в спецификациях | 128 Гб | Не ограничено |
| Ограничение на использование ЦП, кол-во ядер(процессоров) | 1 процессор, количество ядер отсутствует в спецификациях | 24 ядра, 4 процессора | Не ограничено |
| Ограничение на размер БД | 2 Гб | 524 Пб | Не ограничено |
| Наличие поддержки отказоустойчивых кластеров | Отсутствуют | Имеется | Имеется |

Помимо характеристик СУБД, требуется рассмотреть и аппаратно-программные требования программного обеспечения. Результаты рассмотрения находятся в Таблице 11.

Таблица 11 – Аппаратно-программные требования представленных СУБД

| **Наименование СУБД** | **Microsoft Access** | **Microsoft SQL Server Standart** | **PostgreSQL** |
| --- | --- | --- | --- |
| Требования к ЦП | 2-ядерный ЦП с частотой не менее 1,6 ГГц | 64-разрядный или поддерживающий инструкции с частотой не менее 1,4 ГГц | 32-разрядный либо 64-разрядный процессор с частотой не менее 600 МГц |
| Требования  к ОЗУ | 4 Гб | 1 Гб | 192 Мб |
| Требования  к ПЗУ | 4 Гб | 6 Гб | 200 Мб |
| Требования к ПО | Windows 7 и выше с установленным пакетом Microsoft .Net Framework версии 3.5 и выше | Для версии SQL Server 2019 – Windows 10 TH1 ver.1507 или более поздней версии; Windows Server 2016 или более поздней версии | Windows XP SP3 и выше |

Несмотря на то, что СУБД PostgreSQL выигрывает в сравнении характеристик СУБД, изображенных в Таблице 10, для данной работы будет использоваться СУБД Microsoft SQL Server, так как она уже развернута на вычислительном центре института и изучалась в ходе обучения по предмету «Технологии разработки и защиты баз данных».

# Специальный раздел

## 2.1 Архитектурное проектирование

АИС «Расчёт зарплаты» имеет клиент-серверную архитектуру.

В компьютерных технологиях клиент-серверная архитектура предполагает наличие следующих компонентов приложения: клиентское приложение (обычно говорят «тонкий клиент» или терминал), подключенное к серверу, который в свою очередь может быть подключен к серверу базы данных. В качестве сервера может выступать система управления базами данных. Пример клиент-серверной архитектуры показан на рисунке 9.

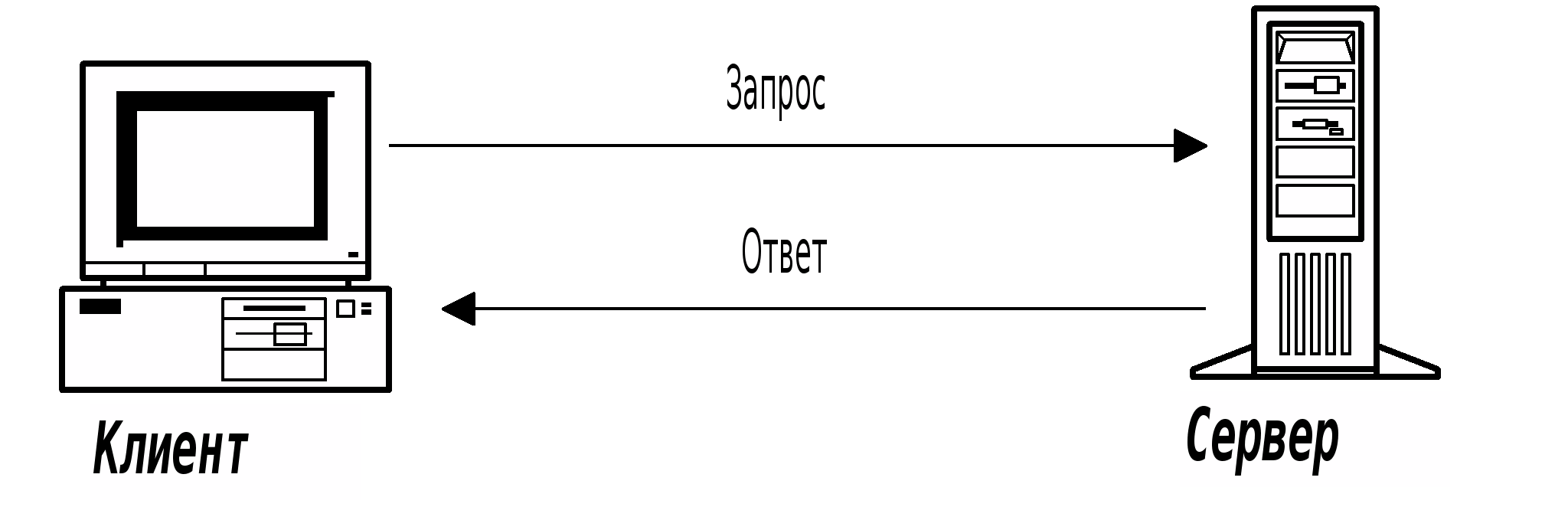


Рисунок 9 – Визуализация клиент-серверной архитектуры

Клиент — это интерфейсный (обычно графический) компонент, который представляет собственно приложение для конечного пользователя.

Сервер базы данных обеспечивает хранение данных. Обычно это стандартная реляционная или объектно-ориентированная СУБД.

Достоинства:

* масштабируемость;
* конфигурируемость — изолированность уровней друг от друга позволяет быстро и простыми средствами переконфигурировать систему при возникновении сбоев или при плановом обслуживании на одном из уровней;
* высокая безопасность;
* высокая надёжность;
* низкие требования к скорости канала (сети) между терминалами и сервером приложений;
* низкие требования к производительности и техническим характеристикам терминалов, как следствие снижение их стоимости.

## 2.2 Проектирование базы данных

Для функционирования информационной системы необходимо разработать базу данных. Для создания реляционной базы данных будем использовать Microsoft SQL Server и систему Microsoft SQL Management Studio. Данная система позволяет администрировать SQL-сервер непосредственно через браузер компьютера. На сегодняшний день данная система пользуется большим спросом у разработчиков, так как предоставляет широкий функционал для работы с базами данных. Для реализации базы данных были созданы следующие таблицы:

* users – хранение данных пользователей;
* baseSalary – хранение типа и суммы зарплаты;
* awards – хранение данных о премировании;
* paynamentMethods – хранение банковских данных сотрудников;
* accurals – хранение данных о выплатах заработной платы;
* timetable – хранение таблицы учета отработанного времени.

Словарь данных и схема базы данных представлены ниже в таблицах 12–17 и рисунке 11.

Таблица 13 – Словарь данных для таблицы baseSalary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **Ключ** | **Описание** |
| userid | Int | NOT | FK Users | ID пользователя |
| type | bool | NOT |  | 0 – ставка 1 - почасовая |
| sum | money | NOT |  | Зарплата |

Таблица 12 – Словарь данных для таблицы Users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **Ключ** | **Описание** |
| id | Int | NOT | PK, Identify(1) | ID пользователя |
| Login | Varchar(50) | NOT |  | Логин пользователя |
| Password | Varchar(50) | NOT |  | Пароль пользователя |
| FIO | Varchar(255) | NOT |  | ФИО сотрудника |
| email | Varchar(50) |  |  | E-mail сотрудника |
| role | Varchar(10) |  |  | Buh – бухгалтер admin – админ  Emp - сотрудник |

Таблица 14 – Словарь данных для таблицы accurals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **Ключ** | **Описание** |
| id | Int | NOT | Identify(1) | ID выплаты |
| userID | Int | NOT | FK Users | ID пользователя |
| sum | Money | NOT |  | Сумма выплаты |
| date | Date | NOT |  | Дата выплаты |

Таблица 15 – Словарь данных для таблицы timeTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **Ключ** | **Описание** |
| id | Int | NOT | Identify(1) | ID |
| userID | Int | NOT | FK Users | ID пользователя |
| workHours | int | NOT |  | Количество отработанных часов |
| date | Date | NOT |  | Дата |

Таблица 16 – Словарь данных для таблицы paynamentMethods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **Ключ** | **Описание** |
| userID | Int | NOT | FK Users | ID пользователя |
| BIK | int | NOT |  | БИК-номер банка |
| INN | int | NOT |  | ИНН банка |
| korr | int | NOT |  | Корреспондентский счет банка |
| AccNum | int | NOT |  | Номер счета сотрудника |

Таблица 17 – Словарь данных для таблицы awards

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Тип** | **NULL** | **ключ** | **Описание** |
| id | Int | NOT | Identify(1) | ID премии |
| userID | Int | NOT | FK Users | ID пользователя |
| Month | int | NOT |  | Месяц назначения премии |
| awardPercent | int | NOT |  | Процент премии |

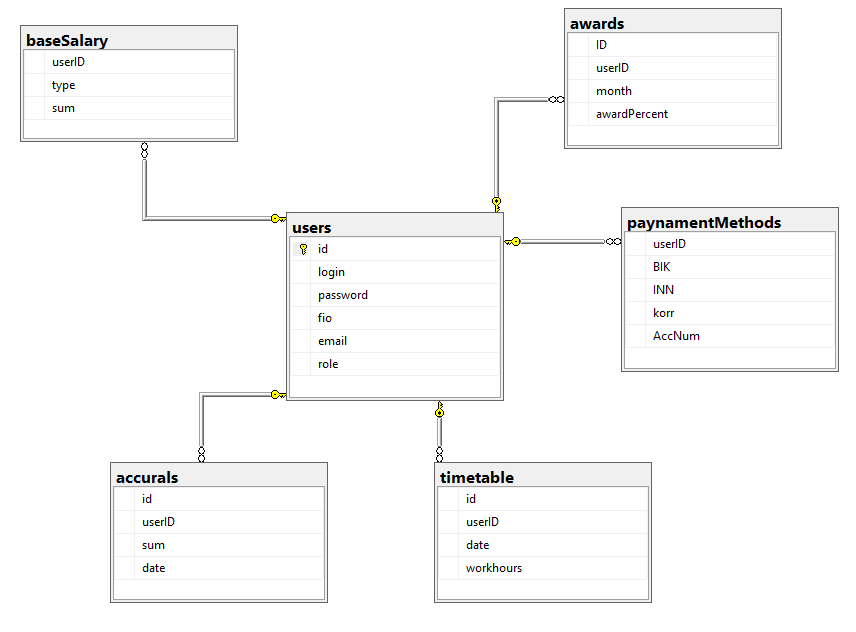


Рисунок 11 – Схема базы данных

## 2.3 Проектирование пользовательского интерфейса

Интерфейс пользователя (UI) - это часть программы, которая находится на виду у пользователя и призвана обеспечивать отображение данных, управление или диалог с пользователем.

Графический интерфейс пользователя (GUU), графический пользовательский интерфейс (GUI) — разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений.

В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в UI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Чаще всего заказчик судит о качестве разработанного программного продукта по интерфейсу. Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом понимается регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи: обмен информацией и координация действий. Диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера. Карты навигации по приложению и спроектированный интерфейс (Wireframe) отображен на рисунках 11 – 23.

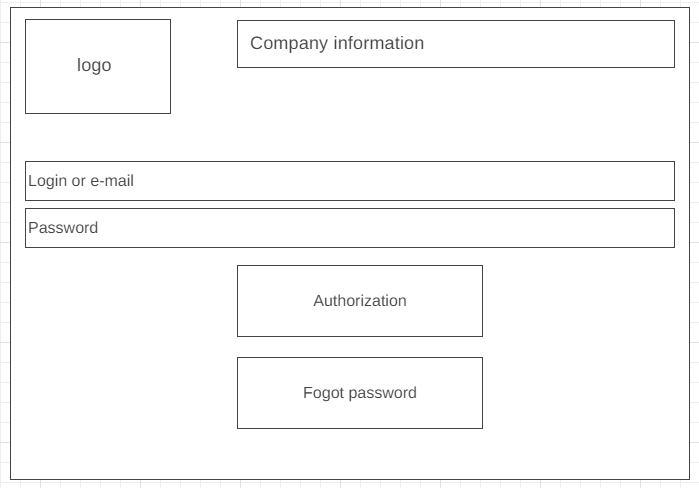


Рисунок 11 – Wireframe окна «Авторизация»

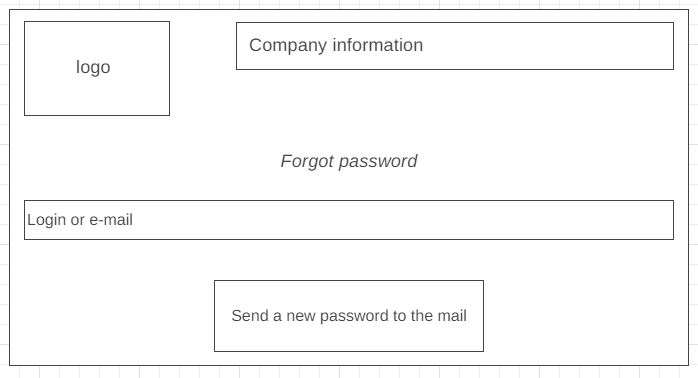


Рисунок 12 – Wireframe окна «Восстановление пароля»

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 – Wireframe окна «Личный кабинет» для роли «Бухгалтер»



Рисунок 14 – Wireframe окна «Личный кабинет» для роли «Администратор»



Рисунок 15 – Wireframe окна «Личный кабинет» для роли «Сотрудник»

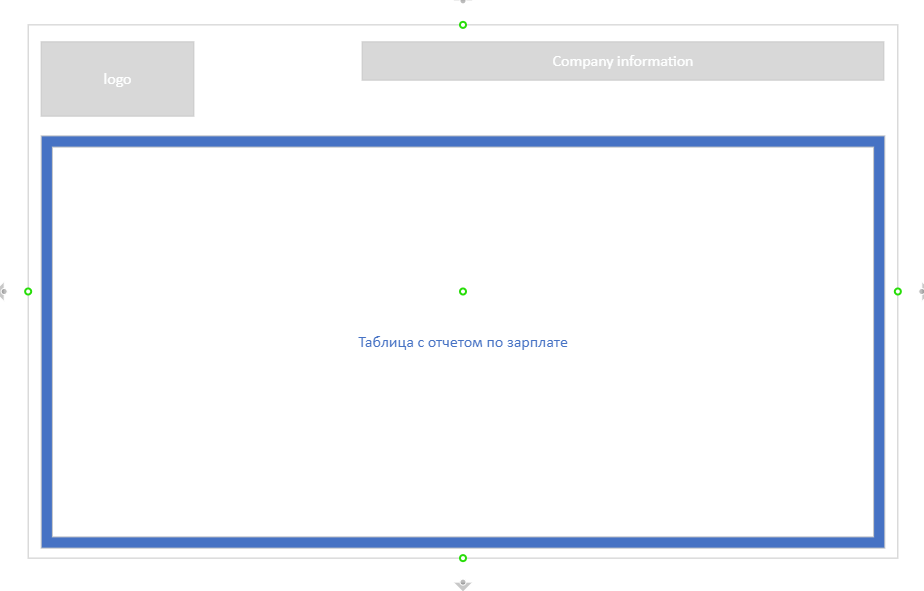


Рисунок 16 – Wireframe окна «Отчеты» для всех ролей



Рисунок 17 – Wireframe окна «Табель рабочего времени» для всех сотрудников

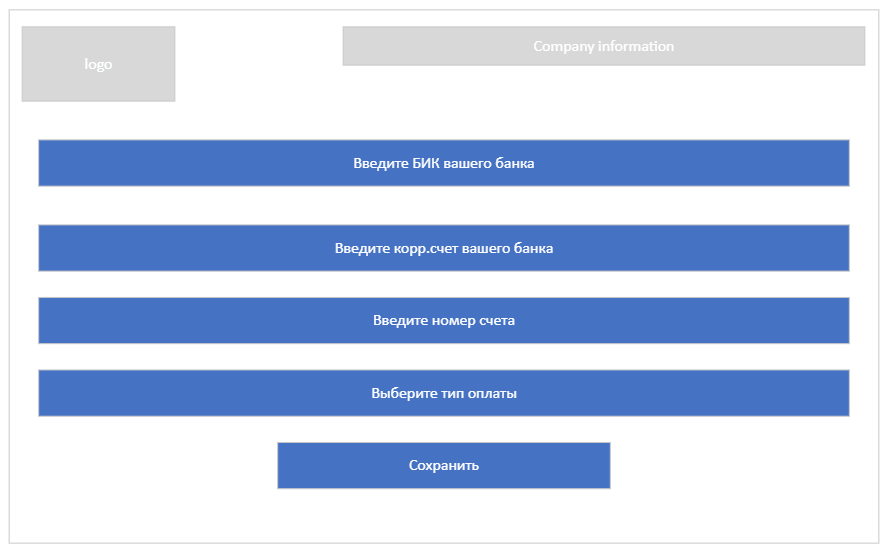


Рисунок 18 – Wireframe окна «Редактирование платежных данных» для всех сотрудников

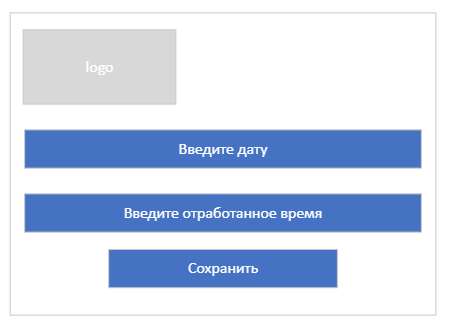


Рисунок 19 – Wireframe окна «Редактирование отработанного времени» для всех сотрудников

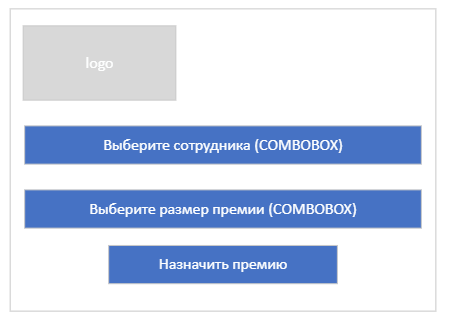


Рисунок 20 – Wireframe окна «Назначение премий» для роли «Администратор» и «Бухгалтер»

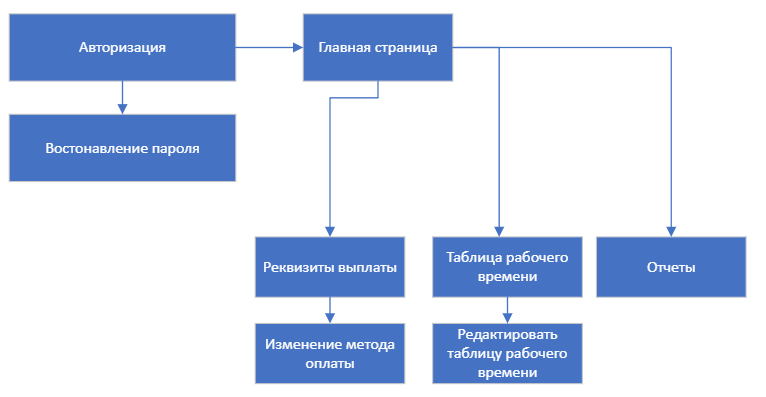


Рисунок 21 – Карта навигации для роли «Сотрудник»



Рисунок 22 - Карта навигации для роли «Бухгалтер»

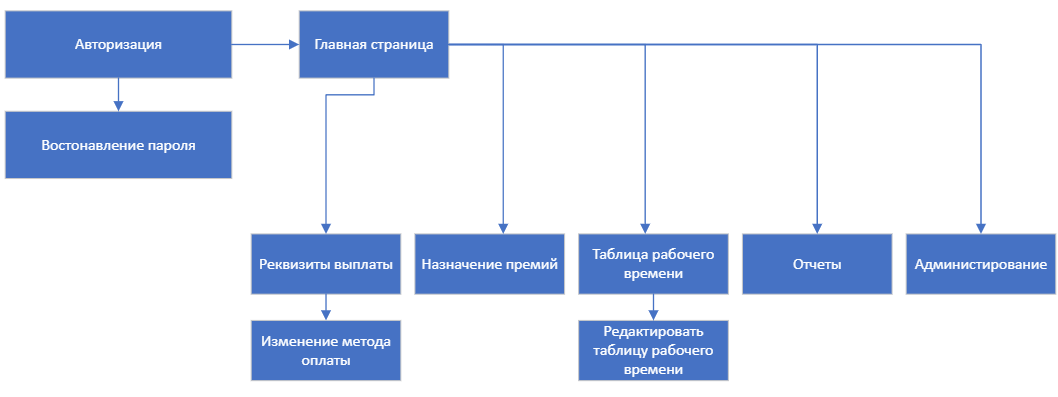


Рисунок 23 - Карта навигации для роли «Администратор»

## 2.4 Проектирование модулей программного обеспечения

Основной задачей проектирования является превращение модели анализа в документы детализированного проектирования, на основе которых реализуется система. Логическая модель проектируемой подсистемы строится на основе технологии Rational и использует основные объектно-ориентированные подходы языка UML.

В процессе проектирования используются нефункциональные требования к системе и ограничения налагаемые на архитектуру, в результате чего модель анализа приобретает новую форму – модель проектирования, которая затем может быть напрямую реализована в виде программного кода.

Для автоматизации всех требований Заказчика, собранных в разделе 1, информационная система должна содержать следующие модули:

* модуль расчёта зарплаты;
* модуль вывода отчёта;
* модуль авторизации;
* модуль ввода информации о сотрудниках;
* модуль управления пользователями.

На основе этих данных можно построить диаграмму деятельности, состояний, последовательностей.

Диаграмма последовательностей — диаграмма, на которой показано разложение некоторой активности на её составные части. Диаграмма последовательности использования личного кабинета пользователя под ролью «Сотрудник» изображена на рисунке 24.

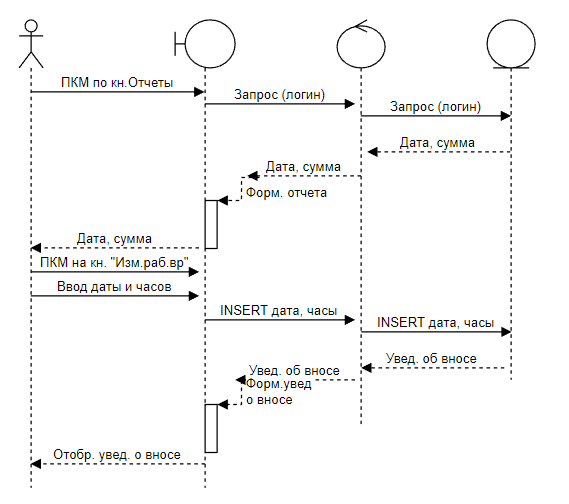


Рисунок 24 – Диаграмма последовательностей

Диаграмма деятельности (англ. activity diagram) — UML-диаграмма, на которой показаны действия, состояния которых описаны на диаграмме состояний. Под деятельностью (англ. activity) понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и отдельных действий англ. action, соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

Диаграмма деятельности показана на рисунке 25.



Рисунок 25 – Диаграмма деятельности

## 2.5 Реализация программного обеспечения

Реализация программного обеспечения – это процесс перевода системной спецификации в работоспособную систему. Итогом реализации приложения является работоспособная информационная система.

Разработка набора элементов информационной системы осуществляется в едином рабочем пространстве.

В качестве основных инструментов для разработки приложения использовались:

* интегрированная среда разработки Visual Studio 2019;
* СУБД Microsoft SQL Server 2018, который обслуживает запросы клиентского приложения и позволяет перенести часть реализуемых задач непосредственно на базу данных, а так же предоставляет возможность получения необходимой пользователю информации в удобном виде.

Код разработанного приложения находится в Приложении А к данной пояснительной записке, скриншоты разработанного интерфейса визуализированны на рисунках 27 –

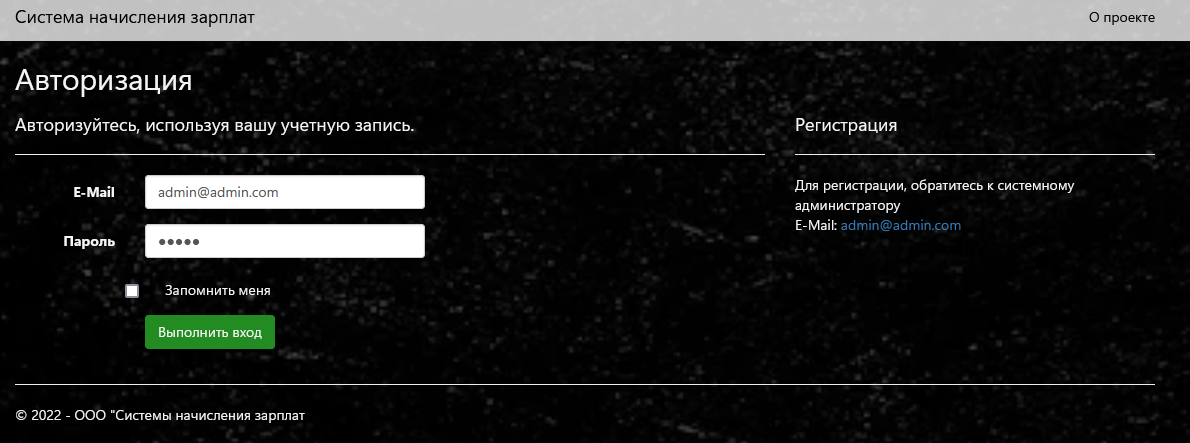


Рисунок 27 - Авторизация

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 28 – Личный кабинет администратора

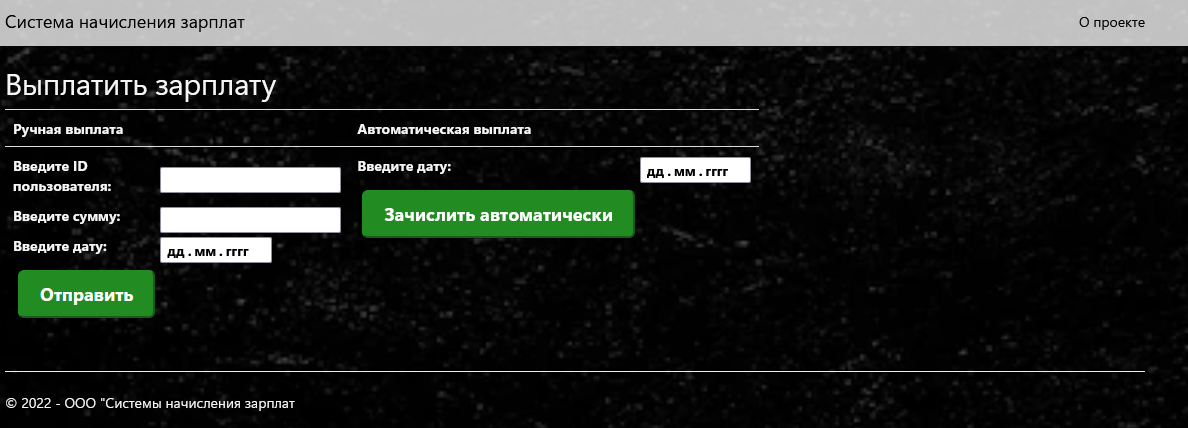


Рисунок 29 – Интерфейс модуля выплаты зарплаты

Изображение выглядит как текст, электроника, снимок экрана

Автоматически созданное описание

Рисунок 29 – Интерфейс модуля отчетов

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 30 – Интерфейс модуля назначения премий

Изображение выглядит как текст, ночное небо

Автоматически созданное описание

Рисунок 30 – Интерфейс модуля работы с таблицами рабочего времени

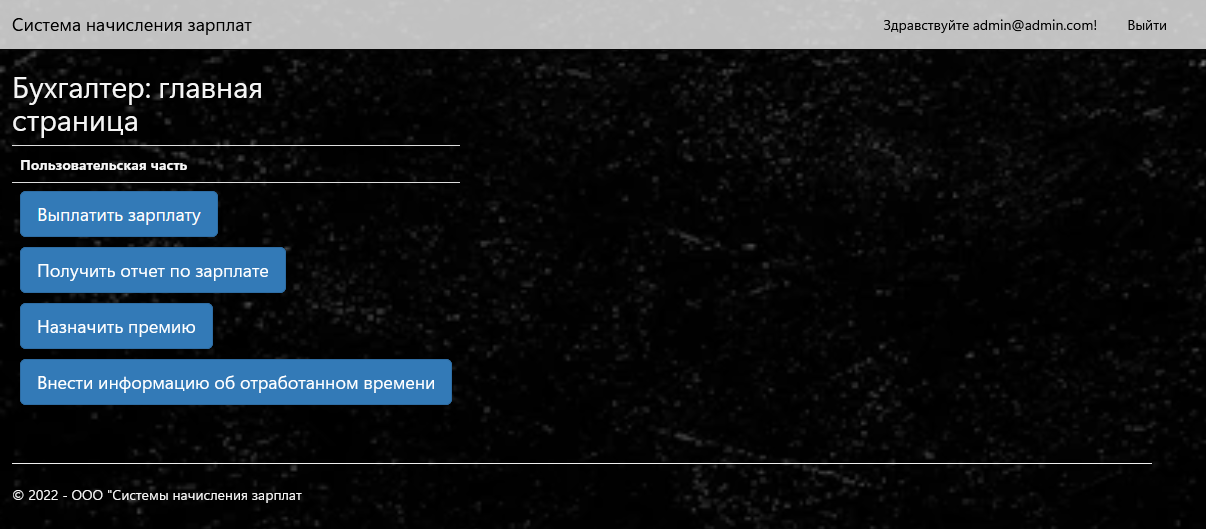


Рисунок 31 – Интерфейс главной страницы под ролью «Бухгалтер»

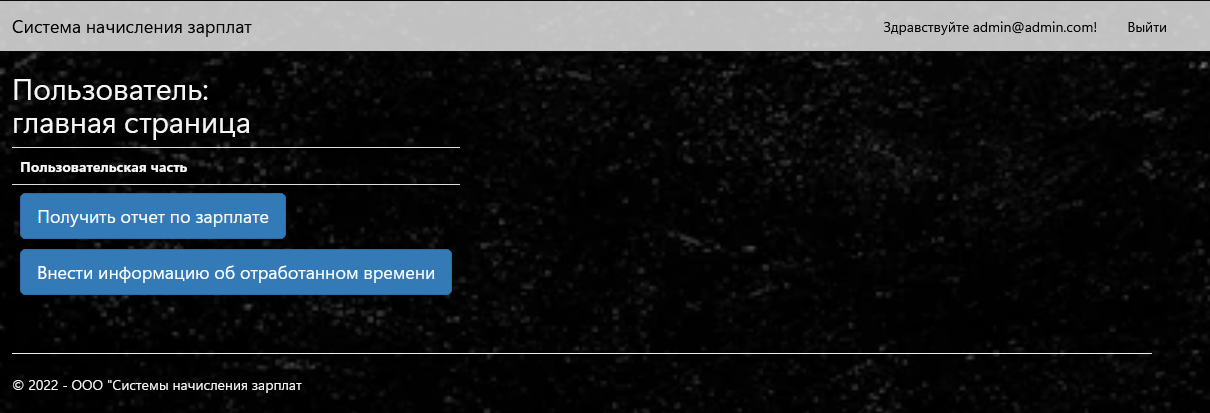


Рисунок 32 – Интерфейс главной страницы под ролью «Сотрудник»

## 2.6 Реализация базы данных

SQL (Structured Query Language) — язык структурированных запросов, является инструментом для выборки и обработки информации, содержащейся в базе данных. SQL - универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных, то есть непосредственно для организации взаимодействия пользователя с базой данных. Если пользователю необходимо получить информацию из базы данных, он запрашивает её у СУБД с помощью SQL. СУБД обрабатывает запрос, находит требуемые данные и посылает их пользователю. Процесс запрашивания данных и получения результата называется запросом к базе данных. SQL используется для реализации всех функциональных возможностей, которые СУБД предоставляют пользователю. К ним относятся:

* организация данных;
* выборка данных;
* обработка данных;
* управление доступом;
* совместное использование данных;
* целостность данных.

В качестве примера взаимодействия приложения с источниками данных можно рассмотреть форму «Начисление заработной платы» под ролью «Бухгалтер».

Первоначальной задачей реализации является визуальное проектирование самой формы, представленной на рисунке 29, средствами Visual Studio 2019.

Для корректной работы с базой данных использовались следующие классы:

* model.edmx – управляющий файл работы с EntityModel;
* DbContext — это сочетание шаблонов "Единица работы" и "Репозиторий". Это внутренний API, который поддерживает инфраструктуру Entity Framework Core и не подчиняется тем же стандартам совместимости, что и общедоступные API;
* Accurals.cs – класс для работы с сущностью в базе данных.

Второй задачей реализации является создание запроса в базу данных, который будет передавать значения из полей формы в таблицу «accurals».

Создавая этот запрос необходимо учитывать передаваемые типы данных. Пример SQL-запроса добавления записи в таблицу представлен на рисунке 33.

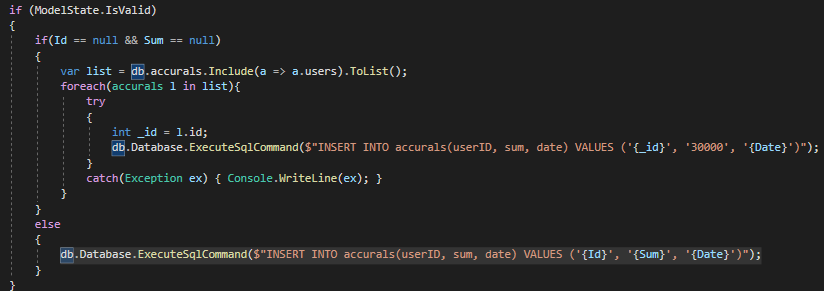


Рисунок 33 – Пример SQL-запроса

Результаты работы формы «Выплатить зарплату» приведены в рисунке 34. Исходя из данных результатов, можно сделать вывод о том, что программный модуль работает корректно.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 34 – Результат работы формы «Выплатить зарплату».   
Визуализация внесения данных в БД

## 2.7 Тестирование приложения

Тестирование приложений, а также разработанных модулей и компонентов является одним из самых важных этапов в реализации АИС.

Тестирование приложения подразумевает 4 этапа:

1. Выбор методов тестирования
2. Создание плана тестирования
3. Разработка тестовых примеров
4. Анализ результатов тестирования

Существует несколько методов тестирования, рассмотрим два из них.

Метод «Белого ящика» используется в случае, когда тестировщиком является человек, который знает все процессы, происходящие в приложении. Как правило, в таких случаях тестировщиком является сам разработчик приложения. Тестирование компонентов приложения на стадии разработки осуществляется в режиме отладки с использованием специального средства Visual Studio 2019. Удобство этого средства тестирования заключается в возможности пошаговой отладки создаваемого приложения в ходе выполнения программы.

Метод «черный ящик» – метод, при котором тестировщик является человеком, не проектировавший данное ПО. Тестировщику дают тестируемое приложение и дают тестовые случаи. В ходе тестирования он должен вносить результаты тестов.

В качестве метода тестирования выбран «Белый ящик», т. к. при тестировании принимается во внимание структура всей программы, что облегчает обнаружение ошибок. План тестирования представляет собой таблицу, в которой указывается действие, ожидаемый и полученный результат. Результаты тестирования приложения методом «Белый ящик» показаны в таблице 18.

Таблица 18 – Test-Case

| **Действие** | **Ожидаемый результат** | **Полученный результат** |
| --- | --- | --- |
| Авторизация на сайте (верный ввод данных пользователя) | Успешная авторизация | Успешная авторизация |
| Авторизация на сайте (неверный ввод данных пользователя) | Ошибка авторизации | Ошибка авторизации |
| Проверка работоспособности ссылок-таблиц на главной странице проекта | Успешный переход | Успешный переход |
| Проверка работоспособности формы «Начислить зарплату» | Данные внесены в БД | Данные внесены в БД |
| Проверка работоспособности вывода всех сущностей из БД | Корректное отображение | Корректное отображение |
| Создание нового пользователя в системе | Пользователь создан и отображается в таблице Пользователи | Пользователь создан и отображается в таблице Пользователи |
| Проверка работы поиска на странице «Пользователи» | Отображен один конкретный пользователь | Отображен один конкретный пользователь |

Модульное тестирование, иногда блочное тестирование или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Первая функция в тестировании проверят открытие страницы, вторая – текст в шапке страницы.

Результат модульного тестирования и тестовый код представлен на рисунках 35–36.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 35 – Код модульного тестирования

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 36 - Сообщение о выполнении

В результате проведенного тестирования мы удостоверились в работе приложения.

# 

# ЗАКЛЮЧЕНИЕ

В данной работе был произведен процесс разработки требований к программному обеспечению, проектирования приложения и его реализации.

В первой главе проведен анализ предметной области, методологий разработки программного обеспечения, существующих решений на рынке по автоматизации данной области. На основе методики интервьюирования и метода опорных точек были разработаны требования к разрабатываемому программному обеспечению.

Во второй главе было спроектировано программное обеспечение, для чего был проведен анализ и проектирование базы данных, проектирование пользовательского интерфейса, проектирование модулей программного обеспечения.

В третьей главе показан процесс реализации программного обеспечения, базы данных, тестирование приложения.

Информационная система написана на языке программирования C#, но стоит отметить, что данная система не отражает всего возложенного на себя функционала, так как программа выдает несколько ошибок, которые пока не удалось исправить, не были реализованы некоторые важные механики для упрощения работы персонала реального предприятия. Таким образом, данную систему можно считать прототипом полностью спроектированной информационной системы для начисления заработной платы, что и было целью данной курсовой работы.

Таким образом, цель работы достигнута полностью.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Bernhard, Rumpe A Framework for Realtime Online Auctions / Rumpe Bernhard, Wimmel Guido. — Текст : непосредственный // Managing Information Technology in a Global Economy. Proceedings of IRMA International Conference. — 2001 : ,Idea Group Publishing, 2001. — С. 13.
2. Paul, Weinberg SQL The Complete Reference / Weinberg Paul, Groff James, Oppel Andrew. — 3-е изд. — New York : The McGraw-Hill Companies, 2010. — 882 c. — Текст : непосредственный.
3. Плещев, В. В. Сравнительный анализ программных средств разработки приложений и баз данных и индивидуализация учебного процесса их изучения / В. В. Плещев. — Текст : непосредственный // Известия УрГЭУ. — 2003. — № 7 — С. 69.
4. Автоаукцион. — Текст : электронный // Википедия - свободная энциклопедия : [сайт]. — URL: ru.wikipedia.org/wiki/Автоаукцион
5. Документация и учебные ресурсы Майкрософт для разработчиков и технических специалистов. — Текст : электронный // Microsoft : [сайт]. — URL: https://docs.microsoft.com/ru-ru/

# ПРИЛОЖЕНИЕ А

(обязательное)

**А.1 Текст программы на языке C#**

AccountController.cs

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

kursach\_pm2Entities DB = new kursach\_pm2Entities();

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

try

{

// Verification.

if (this.Request.IsAuthenticated)

{

// Info.

return this.RedirectToLocal(returnUrl);

}

}

catch (Exception ex)

{

// Info

}

// Info.

return this.View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

try

{

// Verification.

if (ModelState.IsValid)

{

// Initialization.

var loginInfo = this.DB.LoginByUsernamePassword(model.Email, model.Password).ToList();

// Verification.

if (loginInfo != null && loginInfo.Count() > 0)

{

// Initialization.

var logindetails = loginInfo.First();

// Login In.

this.SignInUser(logindetails.email, false);

// ролирование

var selectrole = DB.users.Where(p => p.email == logindetails.email);

var role = selectrole.FirstOrDefault();

if (role.role == "admin")

{

return this.RedirectToLocal("/Admin/Index");

}

if (role.role == "emp")

{

return this.RedirectToLocal("/Emp/Index");

}

if (role.role == "buh")

{

return this.RedirectToLocal("/Buh/Index");

}

return this.RedirectToLocal(returnUrl);

}

else

{

ModelState.AddModelError(string.Empty, "Invalid username or password.");

}

}

}

catch (Exception ex)

{

}

// If we got this far, something failed, redisplay form

return this.View(model);

}

private void SignInUser(string username, bool isPersistent)

{

// Initialization.

var claims = new List<Claim>();

try

{

// Setting

claims.Add(new Claim(ClaimTypes.Name, username));

var claimIdenties = new ClaimsIdentity(claims, DefaultAuthenticationTypes.ApplicationCookie);

var ctx = Request.GetOwinContext();

var authenticationManager = ctx.Authentication;

// Sign In.

authenticationManager.SignIn(new AuthenticationProperties() { IsPersistent = isPersistent }, claimIdenties);

}

catch (Exception ex)

{

// Info

throw ex;

}

}

//

// GET: /Account/VerifyCode

[AllowAnonymous]

public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)

{

// Требовать предварительный вход пользователя с помощью имени пользователя и пароля или внешнего имени входа

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/VerifyCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Приведенный ниже код защищает от атак методом подбора, направленных на двухфакторные коды.

// Если пользователь введет неправильные коды за указанное время, его учетная запись

// будет заблокирована на заданный период.

// Параметры блокирования учетных записей можно настроить в IdentityConfig

var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent: model.RememberMe, rememberBrowser: model.RememberBrowser);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(model.ReturnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неправильный код.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Подтверждение учетной записи", "Подтвердите вашу учетную запись, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

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

}

AddErrors(result);

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// GET: /Account/ConfirmEmail

[AllowAnonymous]

public async Task<ActionResult> ConfirmEmail(string userId, string code)

{

if (userId == null || code == null)

{

return View("Error");

}

var result = await UserManager.ConfirmEmailAsync(userId, code);

return View(result.Succeeded ? "ConfirmEmail" : "Error");

}

//

// GET: /Account/ForgotPassword

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

//

// POST: /Account/ForgotPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))

{

// Не показывать, что пользователь не существует или не подтвержден

return View("ForgotPasswordConfirmation");

}

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);

// var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Сброс пароля", "Сбросьте ваш пароль, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

// return RedirectToAction("ForgotPasswordConfirmation", "Account");

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// GET: /Account/ForgotPasswordConfirmation

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

return View();

}

//

// GET: /Account/ResetPassword

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

return code == null ? View("Error") : View();

}

//

// POST: /Account/ResetPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null)

{

// Не показывать, что пользователь не существует

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);

if (result.Succeeded)

{

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

AddErrors(result);

return View();

}

//

// GET: /Account/ResetPasswordConfirmation

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

return View();

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

// Запрос перенаправления к внешнему поставщику входа

return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/SendCode

[AllowAnonymous]

public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)

{

var userId = await SignInManager.GetVerifiedUserIdAsync();

if (userId == null)

{

return View("Error");

}

var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);

var factorOptions = userFactors.Select(purpose => new SelectListItem { Text = purpose, Value = purpose }).ToList();

return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/SendCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

// Создание и отправка маркера

if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))

{

return View("Error");

}

return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe = model.RememberMe });

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Login");

}

// Выполнение входа пользователя посредством данного внешнего поставщика входа, если у пользователя уже есть имя входа

var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = false });

case SignInStatus.Failure:

default:

// Если у пользователя нет учетной записи, то ему предлагается создать ее

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email = loginInfo.Email });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string returnUrl)

{

if (User.Identity.IsAuthenticated)

{

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

}

if (ModelState.IsValid)

{

// Получение сведений о пользователе от внешнего поставщика входа

var info = await AuthenticationManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user);

if (result.Succeeded)

{

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

try

{

// Setting.

var ctx = Request.GetOwinContext();

var authenticationManager = ctx.Authentication;

// Sign Out.

authenticationManager.SignOut();

}

catch (Exception ex)

{

// Info

}

// Info.

return this.RedirectToAction("Login", "Account");

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

if (\_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

if (\_signInManager != null)

{

\_signInManager.Dispose();

\_signInManager = null;

}

}

base.Dispose(disposing);

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private ActionResult RedirectToLocal(string returnUrl)

{

try

{

// Verification.

if (Url.IsLocalUrl(returnUrl))

{

// Info.

return this.Redirect(returnUrl);

}

}

catch (Exception ex)

{

// Info

}

// Info.

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

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

}

}

AccuralsController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class accuralsController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

// GET: accurals

public ActionResult Index()

{

var accurals = db.accurals.Include(a => a.users);

return View(accurals.ToList());

}

// GET: accurals/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

accurals accurals = db.accurals.Find(id);

if (accurals == null)

{

return HttpNotFound();

}

return View(accurals);

}

// GET: accurals/Create

public ActionResult Create()

{

ViewBag.userID = new SelectList(db.users, "id", "login");

return View();

}

// POST: accurals/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "id,userID,sum,date")] accurals accurals)

{

if (ModelState.IsValid)

{

db.accurals.Add(accurals);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", accurals.userID);

return View(accurals);

}

// GET: accurals/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

accurals accurals = db.accurals.Find(id);

if (accurals == null)

{

return HttpNotFound();

}

ViewBag.userID = new SelectList(db.users, "id", "login", accurals.userID);

return View(accurals);

}

// POST: accurals/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "id,userID,sum,date")] accurals accurals)

{

if (ModelState.IsValid)

{

db.Entry(accurals).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", accurals.userID);

return View(accurals);

}

// GET: accurals/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

accurals accurals = db.accurals.Find(id);

if (accurals == null)

{

return HttpNotFound();

}

return View(accurals);

}

// POST: accurals/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

accurals accurals = db.accurals.Find(id);

db.accurals.Remove(accurals);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

AdminController.cs

using Microsoft.AspNet.Identity;

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class AdminController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

[Authorize]

// GET: Admin

public ActionResult Index()

{

return View();

}

public ActionResult Pay()

{

return View();

}

[HttpPost]

public ActionResult Pay(string Id, string Sum, string Date)

{

if (ModelState.IsValid)

{

if(Id == null && Sum == null)

{

var list = db.accurals.Include(a => a.users).ToList();

foreach(accurals l in list){

try

{

int \_id = l.id;

db.Database.ExecuteSqlCommand($"INSERT INTO accurals(userID, sum, date) VALUES ('{\_id}', '30000', '{Date}')");

}

catch(Exception ex) { Console.WriteLine(ex); }

}

}

else

{

db.Database.ExecuteSqlCommand($"INSERT INTO accurals(userID, sum, date) VALUES ('{Id}', '{Sum}', '{Date}')");

}

}

return RedirectToAction("Index");

}

public ActionResult AddTime()

{

return View();

}

[HttpPost]

public ActionResult AddTime(string Date, string Time)

{

if (ModelState.IsValid)

{

string email = User.Identity.GetUserName();

// INSERT INTO timetable(userID,date, workhours) VALUES ((SELECT id from users where email = 'admin@admin.com'), '06-12-2022', 9)

db.Database.ExecuteSqlCommand($"INSERT INTO timetable(userID,date, workhours) VALUES ((SELECT id from users where email = '{email}'), '{Date}', '{int.Parse(Time)}')");

}

return RedirectToAction("Index");

}

public ActionResult SetAward()

{

return View();

}

[HttpPost]

public ActionResult SetAward(string userID, string month, string percent)

{

if (ModelState.IsValid)

{

db.Database.ExecuteSqlCommand($"INSERT INTO awards(userID, [month], [awardPercent]) VALUES ('{userID}', '{month}', {int.Parse(percent)})");

}

return RedirectToAction("Index");

}

public ActionResult GetMeAccurals()

{

var accurals = from s in db.accurals select s;

string name = User.Identity.GetUserName();

accurals = accurals.Where(s => s.users.email.Contains(name));

return View(accurals.ToList());

}

}

}

AwardsController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class awardsController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

// GET: awards

public ActionResult Index()

{

var awards = db.awards.Include(a => a.users);

return View(awards.ToList());

}

// GET: awards/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

awards awards = db.awards.Find(id);

if (awards == null)

{

return HttpNotFound();

}

return View(awards);

}

// GET: awards/Create

public ActionResult Create()

{

ViewBag.userID = new SelectList(db.users, "id", "login");

return View();

}

// POST: awards/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "ID,userID,month,awardPercent")] awards awards)

{

if (ModelState.IsValid)

{

db.awards.Add(awards);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", awards.userID);

return View(awards);

}

// GET: awards/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

awards awards = db.awards.Find(id);

if (awards == null)

{

return HttpNotFound();

}

ViewBag.userID = new SelectList(db.users, "id", "login", awards.userID);

return View(awards);

}

// POST: awards/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "ID,userID,month,awardPercent")] awards awards)

{

if (ModelState.IsValid)

{

db.Entry(awards).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", awards.userID);

return View(awards);

}

// GET: awards/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

awards awards = db.awards.Find(id);

if (awards == null)

{

return HttpNotFound();

}

return View(awards);

}

// POST: awards/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

awards awards = db.awards.Find(id);

db.awards.Remove(awards);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

BuhController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace WebApplication2.Controllers

{

public class BuhController : Controller

{

// GET: Buh

public ActionResult Index()

{

return View();

}

}

}

EmpController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace WebApplication2.Controllers

{

public class EmpController : Controller

{

// GET: Emp

public ActionResult Index()

{

return View();

}

}

}

ManageController.cs

using System;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

[Authorize]

public class ManageController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public ManageController()

{

}

public ManageController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Manage/Index

public async Task<ActionResult> Index(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Ваш пароль изменен."

: message == ManageMessageId.SetPasswordSuccess ? "Пароль задан."

: message == ManageMessageId.SetTwoFactorSuccess ? "Настроен поставщик двухфакторной проверки подлинности."

: message == ManageMessageId.Error ? "Произошла ошибка."

: message == ManageMessageId.AddPhoneSuccess ? "Ваш номер телефона добавлен."

: message == ManageMessageId.RemovePhoneSuccess ? "Ваш номер телефона удален."

: "";

var userId = User.Identity.GetUserId();

var model = new IndexViewModel

{

HasPassword = HasPassword(),

PhoneNumber = await UserManager.GetPhoneNumberAsync(userId),

TwoFactor = await UserManager.GetTwoFactorEnabledAsync(userId),

Logins = await UserManager.GetLoginsAsync(userId),

BrowserRemembered = await AuthenticationManager.TwoFactorBrowserRememberedAsync(userId)

};

return View(model);

}

//

// POST: /Manage/RemoveLogin

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RemoveLogin(string loginProvider, string providerKey)

{

ManageMessageId? message;

var result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(), new UserLoginInfo(loginProvider, providerKey));

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

message = ManageMessageId.RemoveLoginSuccess;

}

else

{

message = ManageMessageId.Error;

}

return RedirectToAction("ManageLogins", new { Message = message });

}

//

// GET: /Manage/AddPhoneNumber

public ActionResult AddPhoneNumber()

{

return View();

}

//

// POST: /Manage/AddPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> AddPhoneNumber(AddPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Создание и отправка маркера

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), model.Number);

if (UserManager.SmsService != null)

{

var message = new IdentityMessage

{

Destination = model.Number,

Body = "Ваш код безопасности: " + code

};

await UserManager.SmsService.SendAsync(message);

}

return RedirectToAction("VerifyPhoneNumber", new { PhoneNumber = model.Number });

}

//

// POST: /Manage/EnableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> EnableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), true);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

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

}

//

// POST: /Manage/DisableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DisableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), false);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

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

}

//

// GET: /Manage/VerifyPhoneNumber

public async Task<ActionResult> VerifyPhoneNumber(string phoneNumber)

{

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), phoneNumber);

// Отправка SMS через поставщик SMS для проверки номера телефона

return phoneNumber == null ? View("Error") : View(new VerifyPhoneNumberViewModel { PhoneNumber = phoneNumber });

}

//

// POST: /Manage/VerifyPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyPhoneNumber(VerifyPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePhoneNumberAsync(User.Identity.GetUserId(), model.PhoneNumber, model.Code);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.AddPhoneSuccess });

}

// Это сообщение означает наличие ошибки; повторное отображение формы

ModelState.AddModelError("", "Не удалось проверить телефон");

return View(model);

}

//

// POST: /Manage/RemovePhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RemovePhoneNumber()

{

var result = await UserManager.SetPhoneNumberAsync(User.Identity.GetUserId(), null);

if (!result.Succeeded)

{

return RedirectToAction("Index", new { Message = ManageMessageId.Error });

}

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.RemovePhoneSuccess });

}

//

// GET: /Manage/ChangePassword

public ActionResult ChangePassword()

{

return View();

}

//

// POST: /Manage/ChangePassword

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.ChangePasswordSuccess });

}

AddErrors(result);

return View(model);

}

//

// GET: /Manage/SetPassword

public ActionResult SetPassword()

{

return View();

}

//

// POST: /Manage/SetPassword

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SetPassword(SetPasswordViewModel model)

{

if (ModelState.IsValid)

{

var result = await UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.SetPasswordSuccess });

}

AddErrors(result);

}

// Это сообщение означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// GET: /Manage/ManageLogins

public async Task<ActionResult> ManageLogins(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.RemoveLoginSuccess ? "Внешнее имя входа удалено."

: message == ManageMessageId.Error ? "Произошла ошибка."

: "";

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user == null)

{

return View("Error");

}

var userLogins = await UserManager.GetLoginsAsync(User.Identity.GetUserId());

var otherLogins = AuthenticationManager.GetExternalAuthenticationTypes().Where(auth => userLogins.All(ul => auth.AuthenticationType != ul.LoginProvider)).ToList();

ViewBag.ShowRemoveButton = user.PasswordHash != null || userLogins.Count > 1;

return View(new ManageLoginsViewModel

{

CurrentLogins = userLogins,

OtherLogins = otherLogins

});

}

//

// POST: /Manage/LinkLogin

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LinkLogin(string provider)

{

// Запрос перенаправления к внешнему поставщику входа для связывания имени входа текущего пользователя

return new AccountController.ChallengeResult(provider, Url.Action("LinkLoginCallback", "Manage"), User.Identity.GetUserId());

}

//

// GET: /Manage/LinkLoginCallback

public async Task<ActionResult> LinkLoginCallback()

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey, User.Identity.GetUserId());

if (loginInfo == null)

{

return RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

var result = await UserManager.AddLoginAsync(User.Identity.GetUserId(), loginInfo.Login);

return result.Succeeded ? RedirectToAction("ManageLogins") : RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

protected override void Dispose(bool disposing)

{

if (disposing && \_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

base.Dispose(disposing);

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private bool HasPassword()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PasswordHash != null;

}

return false;

}

private bool HasPhoneNumber()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PhoneNumber != null;

}

return false;

}

public enum ManageMessageId

{

AddPhoneSuccess,

ChangePasswordSuccess,

SetTwoFactorSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

RemovePhoneSuccess,

Error

}

#endregion

}

}

paynamentMethods.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class paynamentMethodsController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

// GET: paynamentMethods

public ActionResult Index()

{

var paynamentMethods = db.paynamentMethods.Include(p => p.users);

return View(paynamentMethods.ToList());

}

// GET: paynamentMethods/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

paynamentMethods paynamentMethods = db.paynamentMethods.Find(id);

if (paynamentMethods == null)

{

return HttpNotFound();

}

return View(paynamentMethods);

}

// GET: paynamentMethods/Create

public ActionResult Create()

{

ViewBag.userID = new SelectList(db.users, "id", "login");

return View();

}

// POST: paynamentMethods/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "userID,BIK,INN,korr,AccNum")] paynamentMethods paynamentMethods)

{

if (ModelState.IsValid)

{

db.paynamentMethods.Add(paynamentMethods);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", paynamentMethods.userID);

return View(paynamentMethods);

}

// GET: paynamentMethods/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

paynamentMethods paynamentMethods = db.paynamentMethods.Find(id);

if (paynamentMethods == null)

{

return HttpNotFound();

}

ViewBag.userID = new SelectList(db.users, "id", "login", paynamentMethods.userID);

return View(paynamentMethods);

}

// POST: paynamentMethods/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "userID,BIK,INN,korr,AccNum")] paynamentMethods paynamentMethods)

{

if (ModelState.IsValid)

{

db.Entry(paynamentMethods).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", paynamentMethods.userID);

return View(paynamentMethods);

}

// GET: paynamentMethods/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

paynamentMethods paynamentMethods = db.paynamentMethods.Find(id);

if (paynamentMethods == null)

{

return HttpNotFound();

}

return View(paynamentMethods);

}

// POST: paynamentMethods/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

paynamentMethods paynamentMethods = db.paynamentMethods.Find(id);

db.paynamentMethods.Remove(paynamentMethods);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

timeTables.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

public class timetablesController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

// GET: timetables

public ActionResult Index()

{

var timetable = db.timetable.Include(t => t.users);

return View(timetable.ToList());

}

// GET: timetables/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

timetable timetable = db.timetable.Find(id);

if (timetable == null)

{

return HttpNotFound();

}

return View(timetable);

}

// GET: timetables/Create

public ActionResult Create()

{

ViewBag.userID = new SelectList(db.users, "id", "login");

return View();

}

// POST: timetables/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "id,userID,date,workhours")] timetable timetable)

{

if (ModelState.IsValid)

{

db.timetable.Add(timetable);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", timetable.userID);

return View(timetable);

}

// GET: timetables/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

timetable timetable = db.timetable.Find(id);

if (timetable == null)

{

return HttpNotFound();

}

ViewBag.userID = new SelectList(db.users, "id", "login", timetable.userID);

return View(timetable);

}

// POST: timetables/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "id,userID,date,workhours")] timetable timetable)

{

if (ModelState.IsValid)

{

db.Entry(timetable).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.userID = new SelectList(db.users, "id", "login", timetable.userID);

return View(timetable);

}

// GET: timetables/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

timetable timetable = db.timetable.Find(id);

if (timetable == null)

{

return HttpNotFound();

}

return View(timetable);

}

// POST: timetables/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

timetable timetable = db.timetable.Find(id);

db.timetable.Remove(timetable);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

usersController.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using WebApplication2.Models;

namespace WebApplication2.Controllers

{

[Authorize]

public class usersController : Controller

{

private kursach\_pm2Entities db = new kursach\_pm2Entities();

public ViewResult Index(string sortOrder, string searchString, string searchID)

{

ViewBag.NameSortParm = String.IsNullOrEmpty(sortOrder) ? "name\_desc" : "";

ViewBag.IDSortParm = sortOrder == "id" ? "id\_desc" : "id";

var users = from s in db.users select s;

if (!String.IsNullOrEmpty(searchString))

{

users = users.Where(s => s.fio.Contains(searchString));

}

ViewBag.Message = "Hello world!";

switch (sortOrder)

{

case "name\_desc":

users = users.OrderByDescending(s => s.fio);

break;

case "id":

users = users.OrderBy(s => s.id);

break;

case "id\_desc":

users = users.OrderByDescending(s => s.id);

break;

default:

users = users.OrderBy(s => s.fio);

break;

}

return View(users.ToList());

}

// GET: users/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

users users = db.users.Find(id);

if (users == null)

{

return HttpNotFound();

}

return View(users);

}

// GET: users/Create

public ActionResult Create()

{

ViewBag.id = new SelectList(db.paynamentMethods, "userID", "userID");

return View();

}

// POST: users/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "id,login,password,fio,email,role, img")] users users, HttpPostedFileBase upload)

{

if (ModelState.IsValid)

{

if (upload != null && upload.ContentLength > 0)

{

using (var reader = new System.IO.BinaryReader(upload.InputStream))

{

users.img = reader.ReadBytes(upload.ContentLength);

}

}

db.users.Add(users);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.id = new SelectList(db.paynamentMethods, "userID", "userID", users.id);

return View(users);

}

// GET: users/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

users users = db.users.Find(id);

if (users == null)

{

return HttpNotFound();

}

ViewBag.id = new SelectList(db.paynamentMethods, "userID", "userID", users.id);

return View(users);

}

// POST: users/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку.

// Дополнительные сведения см. в статье https://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "id,login,password,fio,email,role, img")] users users, HttpPostedFileBase upload)

{

try

{

if (ModelState.IsValid)

{

db.Entry(users).State = EntityState.Modified;

if (upload != null && upload.ContentLength > 0)

{

using (var reader = new System.IO.BinaryReader(upload.InputStream))

{

users.img = reader.ReadBytes(upload.ContentLength);

}

db.SaveChanges();

}

else

{

db.Entry(users).Property(m => m.img).IsModified = false;

db.SaveChanges();

}

return RedirectToAction("Index");

}

ViewBag.id = new SelectList(db.paynamentMethods, "userID", "userID", users.id);

}

catch (Exception e)

{

Console.WriteLine(e);

}

return View(users);

}

// GET: users/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

users users = db.users.Find(id);

if (users == null)

{

return HttpNotFound();

}

return View(users);

}

// POST: users/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

users users = db.users.Find(id);

db.users.Remove(users);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Б.1 Дамп SQL-базы данных на языке T-SQL**

USE [kursach\_pm2]

GO

/\*\*\*\*\*\* Object: Table [dbo].[accurals] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[accurals](

[id] [int] IDENTITY(1,1) NOT NULL,

[userID] [int] NOT NULL,

[sum] [money] NULL,

[date] [date] NULL

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[awards] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[awards](

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

[userID] [int] NOT NULL,

[month] [int] NULL,

[awardPercent] [int] NULL

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[baseSalary] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[baseSalary](

[userID] [int] NULL,

[type] [int] NULL,

[sum] [int] NULL

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[paynamentMethods] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[paynamentMethods](

[userID] [int] NOT NULL,

[BIK] [int] NULL,

[INN] [int] NULL,

[korr] [int] NULL,

[AccNum] [int] NULL

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[timetable] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[timetable](

[id] [int] IDENTITY(1,1) NOT NULL,

[userID] [int] NOT NULL,

[date] [date] NOT NULL,

[workhours] [int] NOT NULL

) ON [PRIMARY]

GO

/\*\*\*\*\*\* Object: Table [dbo].[users] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[users](

[id] [int] IDENTITY(1,1) NOT NULL,

[login] [varchar](50) NOT NULL,

[password] [varchar](50) NOT NULL,

[fio] [varchar](255) NOT NULL,

[email] [varchar](50) NULL,

[role] [varchar](10) NOT NULL,

[img] [image] NULL,

CONSTRAINT [PK\_users] PRIMARY KEY CLUSTERED

(

[id] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

ALTER TABLE [dbo].[accurals] WITH CHECK ADD CONSTRAINT [FK\_accurals\_users] FOREIGN KEY([userID])

REFERENCES [dbo].[users] ([id])

GO

ALTER TABLE [dbo].[accurals] CHECK CONSTRAINT [FK\_accurals\_users]

GO

ALTER TABLE [dbo].[awards] WITH CHECK ADD CONSTRAINT [FK\_awards\_users] FOREIGN KEY([userID])

REFERENCES [dbo].[users] ([id])

GO

ALTER TABLE [dbo].[awards] CHECK CONSTRAINT [FK\_awards\_users]

GO

ALTER TABLE [dbo].[baseSalary] WITH CHECK ADD CONSTRAINT [FK\_baseSalary\_users] FOREIGN KEY([userID])

REFERENCES [dbo].[users] ([id])

GO

ALTER TABLE [dbo].[baseSalary] CHECK CONSTRAINT [FK\_baseSalary\_users]

GO

ALTER TABLE [dbo].[paynamentMethods] WITH CHECK ADD CONSTRAINT [FK\_paynamentMethods\_users] FOREIGN KEY([userID])

REFERENCES [dbo].[users] ([id])

GO

ALTER TABLE [dbo].[paynamentMethods] CHECK CONSTRAINT [FK\_paynamentMethods\_users]

GO

ALTER TABLE [dbo].[timetable] WITH CHECK ADD CONSTRAINT [FK\_timetable\_users] FOREIGN KEY([userID])

REFERENCES [dbo].[users] ([id])

GO

ALTER TABLE [dbo].[timetable] CHECK CONSTRAINT [FK\_timetable\_users]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[LoginByUsernamePassword] Script Date: 12.12.2022 17:38:22 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE PROCEDURE [dbo].[LoginByUsernamePassword]

@username varchar(50),

@password varchar(50)

AS

BEGIN

SELECT email, users.password

FROM users

WHERE email = @username

AND users.password = @password

END

GO

USE [master]

GO

ALTER DATABASE [kursach\_pm2] SET READ\_WRITE

GO

# ПРИЛОЖЕНИЕ В

(обязательное)

**В.1 Текст разметки страниц на языке CSHTML**

Addtime.cshtml

@{

ViewBag.Title = "Добавление отработанного времени";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

.btn-primary {

color: #fff;

background-color: forestgreen;

border-color: forestgreen;

padding: 10px 20px;

margin: 5px;

}

\* {

color: whitesmoke;

font-family: 'Segoe UI Variable Text', monospace;

}

.textbox {

color: #000000;

}

</style>

<h2>Добавление отработанного времени</h2>

<form method="post" action="/Admin/AddTime">

<table>

<tr>

<td><p>Введите дату: </p></td>

<td><input type="date" name="Date" class="textbox" /> </td>

</tr>

<tr>

<td><p>Введите отработанное время: </p></td>

<td><input type="text" name="Time" class="textbox" /> </td>

</tr>

<tr>

<td><input type="submit" value="Отправить" class="btn-primary btn-lg" /> </td>

<td></td>

</tr>

</table>

</form>

GetMeAccurals.chtml

@model IEnumerable<WebApplication2.Models.accurals>

@{

ViewBag.Title = "Информация о выплатах";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Информация о выплатах</h2>

<table class="table">

<tr>

<th>

<p>Сумма</p>

</th>

<th>

<p>Дата</p>

</th>

<th>

<p>Кому</p>

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

@Html.DisplayFor(modelItem => item.users.fio)

</td>

</tr>

}

</table>

Admins.chtml

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

.btn-primary {

color: #fff;

background-color: forestgreen;

border-color: forestgreen;

padding: 10px 20px;

margin: 5px;

}

</style>

<div class="row">

<div class="col-md-3 col-sm-6">

<h2>Администратор: главная страница</h2>

<table class="table">

<tr>

<th>

Административная часть

</th>

<th>

Пользовательская часть

</th>

</tr>

<tr>

<th>

<p><a href="/Users/Index" class="btn btn-primary btn-lg">Пользователи</a></p>

<p><a href="/accurals/Index" class="btn btn-primary btn-lg">Выплатаченные зарплаты</a></p>

<p><a href="/timetables/Index" class="btn btn-primary btn-lg">Таблица отработанного времени</a></p>

<p><a href="/paynamentMethods/Index" class="btn btn-primary btn-lg">Платежные данные</a></p>

<p><a href="/awards/Index" class="btn btn-primary btn-lg">Премии</a></p>

</th>

<th>

<p><a href="/Admin/Pay" class="btn btn-primary btn-lg">Выплатить зарплату</a></p>

<p><a href="/Admin/GetMeAccurals" class="btn btn-primary btn-lg">Получить отчет по зарплате</a></p>

<p><a href="/Admin/SetAward" class="btn btn-primary btn-lg">Назначить премию</a></p>

<p><a href="/Admin/AddTime" class="btn btn-primary btn-lg">Внести информацию об отработанном времени</a></p>

</th>

</tr>

</table>

</div>

</div>

Pay.chstml

@{

ViewBag.Title = "Выплата зарплаты";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

.btn-primary {

color: #fff;

background-color: forestgreen;

border-color: forestgreen;

padding: 10px 20px;

margin: 5px;

}

\* {

color: whitesmoke;

font-family: 'Segoe UI Variable Text', monospace;

}

.textbox{

color: #000000;

}

</style>

<h2>Выплатить зарплату</h2>

<div class="row">

<div class="col-md-3 col-sm-6">

<table class="table">

<tr>

<th>

Ручная выплата

</th>

<th>

Автоматическая выплата

</th>

</tr>

<tr>

<th>

<form method="post" action="/Admin/Pay">

<table>

<tr>

<td><p>Введите ID пользователя: </p></td>

<td><input type="text" name="Id" class="textbox"/> </td>

</tr>

<tr>

<td><p>Введите сумму: </p></td>

<td><input type="text" name="Sum" class="textbox"/> </td>

</tr>

<tr>

<td><p>Введите дату: </p></td>

<td><input type="date" name="Date" class="textbox"/> </td>

</tr>

<tr>

<td><input type="submit" value="Отправить" class="btn-primary btn-lg"/> </td>

<td></td>

</tr>

</table>

</form>

</th>

<th>

<form method="post" action="/Admin/Pay">

<table>

<tr>

<td><p>Введите дату: </p></td>

<td><input type="date" name="Date" class="textbox" /> </td>

</tr>

<tr>

<td><input type="submit" value="Зачислить автоматически" class="btn-primary btn-lg" /> </td>

<td></td>

</tr>

</table>

</form>

</th>

</tr>

</table>

</div>

</div>

Setaward.chstml

@{

ViewBag.Title = "Назначение премий";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<style>

.btn-primary {

color: #fff;

background-color: forestgreen;

border-color: forestgreen;

padding: 10px 20px;

margin: 5px;

}

\* {

color: whitesmoke;

font-family: 'Segoe UI Variable Text', monospace;

}

.textbox {

color: #000000;

}

</style>

<h2>Назначение премий</h2>

<form method="post" action="/Admin/SetAward">

<table>

<tr>

<td><p>Введите ID пользователя: </p></td>

<td><input type="text" name="userID" class="textbox" /> </td>

</tr>

<tr>

<td><p>Введите месяц назначение (номер месяца): </p></td>

<td><input type="text" name="month" class="textbox" /> </td>

</tr>

<tr>

<td><p>Введите процент премии: </p></td>

<td>

<select class="textbox" name="percent">

<option disabled>Выберите</option>

<option value="30">30</option>

<option selected value="15">15</option>

<option value="10">10</option>

</select>

</td>

</tr>

<tr>

<td><input type="submit" value="Отправить" class="btn-primary btn-lg" /> </td>

<td></td>

</tr>

</table>

</form>