МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем
Кафедра «Информационные технологии»
Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
по дисциплине «Объектно-ориентированное программирование»

на тему: «*Windows Form* приложение для учёта успеваемости студентов ВУЗа»

Исполнитель: студент гр. ИТП-22
Ялченко М.Д.

Руководитель: доцент
Курочка К.С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____
Оценка работы: _____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время проблемы, связанные с совершенствованием и объективизацией системы оценивания учебной деятельности студентов, являются одним из центральных направлений развития в системе образования. Многие традиционные методы оценки знаний, умений и навыков не соответствуют современным требованиям, предъявляемым к качеству образования. Решение данных проблем оказывается связано с обеспечением удобства использования системы учёта успеваемости всеми участниками образовательного процесса, в первую очередь преподавателями и студентами. Для удобной организации основных видов педагогической деятельности требуется использование информационных технологий автоматизации процессов, которые на данный момент крайне редко применяются в образовательной практике.

С одной стороны, использование автоматизации в учебном процессе при организации учёта текущей успеваемости необходимо и всеми признано, с другой стороны те решения, которые сейчас используются в вузах, в основном ориентированы на бумажные носители и крайне неудобны в работе. В связи с этим ставится задача реализации этих документов с возможностью автоматизации трудоёмких процессов. В отечественных вузах рейтинг практически не играет роли (не влияет на трудоустройство, не оказывает влияния на перспективы дальнейшего обучения), однако рейтинговый показатель дисциплины важен для управления учебной работой, как со стороны преподавателя, так и для самоорганизации студентов. При этом важно, чтобы журнал текущей успеваемости был доступен студентам в режиме просмотра для самоуправления собственной учебной деятельности. Помимо этого, журнал должен быть удобен для работы преподавателя и при необходимости для контроля со стороны администрации. С этой точки зрения одним из наиболее удобных вариантов является реализация системы автоматизации учёта успеваемости студентов по балльно-рейтинговой системе с хранением информации в специализированной базе данных, когда всем участникам образовательного процесса открывается доступ к необходимой информации.

1 ТЕХНИЧЕСКИЕ ПОДХОДЫ К ОРГАНИЗАЦИИ УЧЁТА ТЕКУЩЕЙ УСПЕВАЕМОСТИ СТУДЕНТОВ

1.1 Система управления базами данных

Система управления базами данных (СУБД) – комплекс программ, позволяющих создать базу данных, с последующей манипуляцией её данными (вставлять, обновлять, удалять, выбирать данные). Система должна обеспечивать безопасность, надёжность хранения и целостность данных, а также предоставлять средства для администрирования базы данных [1, с. 39].

База данных – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняется в соответствии с правилами средств моделирования данных [8].

Основные функции СУБД:

– Управление данными во внешней памяти (на дисках);

– Управление данными в оперативной памяти с использованием дискового кэша;

– Журнализация изменений, резервное копирование и восстановление баз данных в случае сбоев;

– Поддержка языков базы данных.

– Современная СУБД должна содержать следующие компоненты:

– Ядро – отвечает за управление данными во внешней и оперативной памяти и журнализацию;

– Процессор языка базы данных – обеспечивает оптимизацию запросов на извлечение и изменение данных и создание машинно-независимого исполняемого кода;

– Подсистема поддержки времени исполнения – интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.

– Сервисные программы – обеспечивают ряд дополнительных возможностей по обслуживанию информационной системы.

СУБД бывают локальные и распределённые. Локальные СУБД – это СУБД, все части которой размещаются на одном компьютере. У распределённых СУБД части могут размещаться на двух и более компьютерах.

По способу доступа к базе данных СУБД делятся на:

– Файл-серверные – все файлы располагаются централизованно на файл-сервере;

– Клиент-серверная СУБД располагается вместе с базой данных и осуществляет доступ к базе данных в многопользовательском режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно;

– Встраиваемая СУБД – это СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки.

1.2 Объектно-реляционная модель данных

Реляционная модель данных – совокупность данных, состоящая из набора двумерных таблиц. Реляционная модель является удобной и наиболее привычной формой представления данных, так в настоящее время эта модель является фактически стандартом, на которой ориентируются практически все современные коммерческие базы данных.

Объектная модель данных – такая модель данных, которая основывается на использовании объектно-ориентированных языков программирования. В частности, объектные базы данных создаются за счёт комбинирования возможностей баз данных с функциональностью объектно-ориентированных языков программирования.

Объектно-реляционная модель объединяет реляционную и объектную модели данных. Объекты, классы и наследование реализованы в структуре базы данных и языке запросов.

1.3 СУБД *MySQL*

MySQL – свободная реляционная система управления базами данных. Она базируется на языке *SQL*, в следствии чего поддерживает множество возможностей этого языка.

Преимущества *MySQL*:

– простота в использовании. *MySQL* достаточно легко инсталлируется, а наличие множества плагинов и вспомогательных приложений упрощает работу с базами данных;

– обширный функционал. Система *MySQL* обладает практически всем необходимым инструментарием, который может понадобиться в реализации практически любого проекта;

– безопасность. Система изначально создана таким образом, что множество встроенных функций безопасности в ней работают по умолчанию;

– расширяемая система встроенных языков программирования и поддержку загрузки *C*-совместимых модулей;

– универсальность. *MySQL* работает на ОС семейств *Windows*, *Linux*, *Unix*, *Solaris* и других. Также СУБД имеет *API* для большинства популярных языков программирования: *C* и *C++*, *PHP*, *Python*, *Ruby*, *Java* и других.

Особенности *MySQL*:

Эта система считается относительно дешевой, по сравнению с другими платными СУБД. Она хорошо масштабируется, считается довольно гибкой и простой в использовании. Плюс, изначально система разрабатывалась для управления большими базами данных. Поэтому сейчас она подходит для промышленной эксплуатации, если смотреть с точки зрения скорости работы. Неважно, выполняете ли вы тяжёлую бизнес-аналитику или нужно хранить большие объемы данных электронной коммерции.

Гибкость обеспечена большим количеством вариантов таблиц для хранения данных. Можно выбрать, например, таблицы типа *MyISAM* – поддерживающие полнотекстовый поиск. Также таблицы типа *InnoDB*, которые поддерживают транзакции на уровне отдельных записей. То есть, вы спокойно выбираете то, что подходит именно под ваш проект.

Плюс система поддерживает множество графических интерфейсов – *WorkBench, SequelPro, DBVisualizer* и *Navicat DB*. Какие-то из них доступны только для определенной ОС, какие-то коммерческие. Но в любом случае есть возможность выбрать собственный комфортный формат.

По поводу лицензии, если вы захотите поменять что-то в коде – систему можно распространять с помощью доступов *GNU GPL*, либо под собственной коммерческой лицензией. Если какая-то программа включает в себя исходные коды *MySQL*, то и она должна распространяться по лицензии *GPL*. Так что, если вы не хотите открывать свои исходные данные кода, лучше воспользоваться лицензией. К тому же она дает качественную сервисную поддержку.

1.4 Платформа *.Net Framework* и язык программирования *C#*

Платформа *.Net Framework* – технология, поддерживающая создание и выполнение веб-служб и приложений *Windows*.

Платформа *.Net* предназначена для:

– обеспечения согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределённого в Интернете, либо для удалённого выполнения;

– для разработки приложений на различных языках программирования, поддерживаемых платформой *.Net*;

– предоставления среды выполнения кода, в которой гарантируется безопасное выполнение кода, исключаются проблемы с производительностью сред выполнения скриптов или интерпретируемого кода.

Платформа *.Net Framework* состоит из общеязыковой среды выполнения (среды *CLR*) и библиотеки классов *.Net Framework*. Базисом платформы *.Net Framework* является среда *CLR*. *CLR* – это среда выполнения, которая подходит для разных языков программирования. Основные возможности *CLR* (управление

памятью, загрузка сборок, безопасность, обработка исключений, синхронизация) доступны в любых языках программирования, использующих эту среду. Фактически во время выполнения программы в среде *CLR* неизвестно, на каком языке программирования написан исходный код. А это значит, что можно выбрать любой язык программирования, который позволяет проще решить конкретную задачу [3, с. 28]. Код, который обращается к среде выполнения – безопасный код (*C#*), а код, который не обращается к среде выполнения – небезопасный код (*C++*).

Библиотека классов платформы *.Net Framework* представляет собой коллекцию типов, которые тесно интегрируются со средой *CLR*. Библиотека классов является объектно-ориентированной.

Поскольку библиотека классов является объектно-ориентированной, типы *.Net Framework* позволяет решать типовые задачи программирования, включая работу со строками, сбор данных, подключение к базам данных и доступ к файлам. Библиотека классов используется для разработки следующих типов приложений и служб:

– консольные приложения;

– приложения с графическим интерфейсом для семейства операционных систем *Windows* (*Windows Forms*);

– приложения *Windows Presentation Foundation* (*WPF*);

– приложения *ASP.NET*;

– службы *Windows*;

– сервисноориентированные приложения, использующие Windows Communication Foundation (*WCF*);

– приложения, поддерживающие бизнес-процессы *Windows Workflow Foundation* (*WF*).

C# (*C Sharp*) – объектно-ориентированный и типобезопасный язык программирования. *C#* был разработан для платформы *.Net*. Он относится к семейству *C*-подобных языков программирования, то есть его синтаксис схож с синтаксисом таких языков программирования как *C, C++, Java*.

В *C#* присутствует огромное обилие функций, позволяющих создавать надёжные и устойчивые приложения. Сборка мусора – в отличие от *C++* автоматически освобождает память, занятую недоступными объектами. Типы, допускающие значение *null*, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений – позволяет надёжно и безопасно обнаружить исключение и обработать его. Лямбда-выражения поддерживают приёмы функционального программирования. *LINQ* – удобный язык запросов к данным из любого источника. В *C#* действует единая система типов. Все типы, включая примитивы, такие как *int, double, char, string, decimal*, наследуют от одного базового типа *object*. Это позволяет всем типам иметь базовый функционал, также значения любого типа можно хранить, передавать и обрабатывать

схожим образом. Кроме того, *C#* поддерживает как определяемые пользователем ссылочные типы, так и типы значений. *C#* поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. Также *C#* предоставляет итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

В *C#* особое внимание уделяется управлению версиями для обеспечения совместимости программ и библиотек при их изменении. Вопросы управления версиями существенно повлияли на такие аспекты разработки *C#*, как раздельные модификаторы *virtual* и *override*, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Программы, написанные на *C#*, выполняются в *.Net*, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (*CLR*) и набор библиотек классов.

Исходный код, написанный на языке *C#* компилируется в промежуточный язык (*IL*), который по своей структуре напоминает язык Ассемблера. Код на языке *IL* и ресурсы, в том числе растровые изображения и строки, сохраняются в сборке, обычно с расширением *.dll*. Сборка содержит метаданные с информацией о типах, версии и региональных параметрах для этой сборки.

При выполнении программы сборка загружается в среду *CLR*. Среда *CLR* выполняет *JIT*-компиляцию из кода на языке *IL* в инструкции машинного кода. Среда *CLR* также выполняет автоматическую сборку мусора, обработку исключений и управление ресурсами.

.Net также включает библиотеки, поддерживающие множество различных рабочих нагрузок. Они упорядочены по пространствам имён, которые предоставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа *XML*, от платформ веб-приложений до элементов управления *Windows Forms*[4].

1.5 Технология *Windows Forms*

Windows Forms – это платформа пользовательского интерфейса для создания классических приложений *Windows*. Она обеспечивает один из самых эффективных способов создания классических приложений с помощью визуального конструктора в *Visual Studio*. Такие функции, как размещение визуальных элементов управления путем перетаскивания, упрощают создание классических приложений.

В *Windows Forms* можно разрабатывать графически сложные приложения, которые просто развертывать, обновлять, и с которыми удобно работать как в

автономном режиме, так и в сети. Приложения *Windows Forms* могут получать доступ к локальному оборудованию и файловой системе компьютера, на котором работает приложение.

Предусмотрено множество элементов управления, которые можно добавлять в формы. Например, элементы управления могут отображать текстовые поля, кнопки, раскрывающиеся списки, переключатели и даже веб-страницы. Если предусмотренные элементы управления не подходят для ваших целей, в *Windows Forms* можно создавать собственные пользовательские элементы управления с помощью класса *UserControl*.

Преимущества *Windows Forms*:

– производительность для приложений с простой графикой.
WPF, *DirectX* и *OpenGL* хороши только там, где есть анимации прозрачность и *3D*;

– в *Windows Forms* шрифт привычный для большинства пользователей, в *WPF* расплывчатый, ибо у *DirectX* свой "движок" для отрисовки шрифтов;

– На данный момент существует огромное множество готовых элементов управления, которые можно купить либо использовать бесплатно;

– простые проекты, написанные на *Windows Forms*, можно довольно легко перенести на другую операционную систему, если на ней установлен *.Net Framework* нужной модели, на котором написан ваш проект.

Однако стоит учитывать, что в основе технологии лежат нативные элементы управления *Windows*, их внешний вид сложно изменить, и типовое приложение выглядит скромно (обычно для преодоления этих недостатков используются специальные сторонние библиотеки наподобие *Infragistics* или *DevExpress*).

1.6 Шаблоны проектирования, используемые для реализации информационной системе

Шаблоны проектирования – представляют определённый способ написания программного кода для решения часто встречающихся проблем проектирования [6].

При написании программ программист довольно часто сталкивается с некоторым набором задач, которые, вне зависимости от того какая это программа, имеют общий шаблон написания кода. Для того, чтобы эффективно решить такие задачи необходимо применить соответствующие шаблоны проектирования. Такой подход имеет ряд плюсов: нет необходимости ничего придумывать, при проектировании совершается меньше просчётов, используются типовые унифицированные решения, так как все скрытые проблемы в них уже давно найдены.

Шаблоны бывают трёх видов: порождающие, структурные, поведенческие. Порождающие шаблоны предоставляют механизмы инициализации, позволяя создавать объекты удобным способом. Структурные шаблоны определяют

отношения между классами и объектами, позволяя им работать совместно. Поведенческие шаблоны используются для того, чтобы упростить взаимодействие между сущностями.

Одним из наиболее часто используемых шаблонов при работе с базами данных является шаблон «Репозиторий». Он позволяет абстрагироваться от конкретных подключений к базам данных, с которыми работает программа, и является промежуточным звеном между классами и остальной программой.

Так как приложение работает с базой данных, необходимо позаботиться о создании постоянного подключения, так как процесс подключения к БД довольно ресурсозатратный. Для решения этой проблемы можно использовать шаблон «Одиночка». Этот шаблон гарантирует создание только одного экземпляра для определённого класса.

1.7 Технологии для реализации приложения

Для реализации приложения будет создан автоматизированный программный комплекс на платформе *.NET Framework*, написанный на языке программирования *C#*. В качестве источника данных будет принята СУБД *MySQL*. Графический интерфейс приложения будет создан с помощью технологии *Windows Forms*.

Язык программирования *C#* обладает таким достоинствами как: статическая типизация, события, делегаты, свойства, *LINQ* (*Language Integrated Query* – язык интегрированных запросов), обобщения.

MySQL является бесплатной объектно-реляционной системой управления базами данных. Также данная СУБД поддерживает базы данных неограниченного размера, что является большим плюсом, так как учёт успеваемости студентов явно подразумевает наличие большого массива данных.

2 АРХИТЕКТУРА ПРИЛОЖЕНИЯ УЧЁТА УСПЕВАЕМОСТИ СТУДЕНТОВ ВУЗА

2.1 Архитектура разрабатываемого приложения

Любое приложение, работающее с источником данных (база данных, *JSON*-файл, *XML*-файл и т.д.), должно выполнять следующие действия:

– считать данные из источника данных;

– произвести необходимую обработку данных;

– представить данные пользователю;

– при необходимости отредактировать данные;

– обновить данные в источнике.

Источником данных разрабатываемого приложения является база данных и, соответственно, приложение должно иметь эффективный и удобный функционал для чтения и обработки данных. За обеспечения такого функционала отвечает слой *Object-relational mapping* (*ORM*). *ORM* создает слой между реляционными базами данных и объектно-ориентированными языками программирования без необходимости написания *SQL*-запросов. Отображение стандартизирует интерфейсы, сокращая количество шаблонов и ускоряя время разработки.

Технология преобразует множество состояний и кодов, создавая структурированную карту, которая помогает разработчикам лучше ориентироваться в структуре базы данных.

Отображение объясняет, как объекты связаны с разными таблицами, используя эту информацию для преображения данных между ними. ORM генерирует код SQL для реляционной базы данных в ответ на изменения, которые приложение вносит в объект данных. Сопоставление ORM будет управлять потребностями приложения в данных, избавляя от написания низкоуровневого кода.

Для корректной обработки моделей приложение должно содержать слой, ответственный за бизнес-логику. Такой слой должен работать непосредственно с *ORM* слоем, считывая, редактируя и записывая данные в базу данных, производить необходимую обработку считанных данных и предоставить их графическому интерфейсу. Такой слой является своего рода прослойкой между графическим интерфейсом и моделями, что позволяет отделить логику приложения от представления.

Графический интерфейс должен отвечать только за вывод данных на экран пользователя. Он должен быть прост и интуитивно понятен и должен иметь весь необходимый функционал для удобной работы пользователя.

Схема архитектуры разрабатываемого приложения представлена на рисунке 2.1.

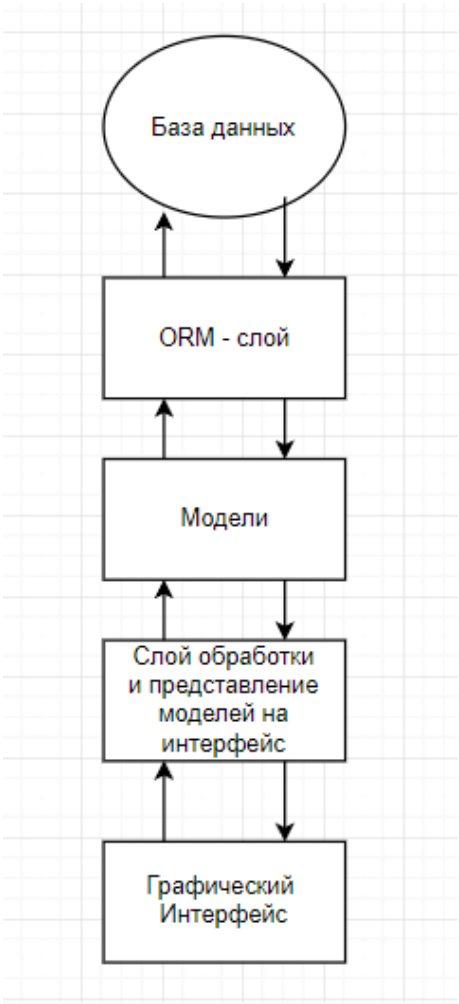

Рисунок 2.1 – Схема архитектуры разрабатываемого приложения

Главное преимущество данной архитектуры – масштабируемость. Каждый компонент отвечает только за свой функционал, что позволяет в будущем легко модернизировать приложение или исправить неполадку, в случае возникновения таковой.

2.2 Структура базы данных

На основании технического задания можно сделать вывод, что приложению необходимо работать с большим количеством данных. Для решения проблемы хранения большого объёма данных используется СУБД *MySQL*. Создание базы данных и обращение к её элементам осуществляется при помощи *SQL*-запросов.

Для решения поставленной задачи разработана база данных, с использованием взаимосвязанных нормализованных таблиц. Каждая таблица описывает

определённую сущность. При проектировании базы данных использованы различные типы данных: *int, bool, varchar(), date*. Для добавления каждому объекту уникального индекса, использованы автоинкриментированные первичные ключи, в название которых для очевидности добавлена приставка «*Id*». Для взаимосвязи таблиц использованы связи «один к одному» и «один ко многим» при помощи внешних ключей.

Каждая таблица – описание списка конкретных объектов, а каждая строка в таблице – конкретный объект. Каждый столбец таблицы – определённый атрибут объекта.

В университете имеются факультеты, специальности. К специальностям относятся различные группы. В каждой группе обучается определенной количество студентов.

Приложение должно обеспечивать возможность просмотра всех факультетов, специальностей, групп, студентов и испытаний. Каждая специальность относиться к определенному факультету, а каждая группа относиться к определенной специальности. В каждой группе обучаются студенты.

Студенты могут сдавать несколько видов испытания: зачеты, экзамены, курсовые работы. Набор испытаний зависит от специальности и курса. В результате сдачи испытания каждому студенту выставляется оценка. Студент может не сдать испытание, получив отрицательную оценку. Такое испытание студент может пересдать.

Все данные, используемые приложением, хранятся в базе данных в виде следующих таблиц:

– *Faculties*;
– *Specialties*;
– *StudentGroups*;
– *Students*;
– *Teachers*;
– *Exams*;
– *PassExams*;
– *Credits*;
– *PassCredit*;
– *CourseWorks*;
– *PassCourseWorks*;
– *TeacherLogins*;
– *StudentLogins*;
– *Admins*.

Таблица *Faculties* содержит следующие столбцы: *id* – уникальный идентификатор факультета; *FullName* – полное название факультета; *ShortName* – сокращённое название факультета.

Таблица *Specialties* содержит следующие столбцы: *id* – уникальный идентификатор специальности; *FullName* – название специальности, должно быть уникальным; *FacultId* – ссылка на факультет из таблицы *Faculties*, которому принадлежит данная специальность.

Таблица *StudentGroups* содержит следующие столбцы: *id* – уникальный идентификатор группы; *GroupName* – название группы; *Course* – номер курса, на котором обучаются все студенты данной группы; *SpecId* – ссылка на специальность из таблицы *Specialties*, по которой обучаются все студенты данной группы.

Таблица *Students* содержит следующие столбцы: *id* – уникальный идентификатор студента; *Name* – имя студента; *Surname* – фамилия студента; *Patronymic* – отчество студента; *GroupId* – ссылка на группу из таблицы *StudentGroups*, которой принадлежит студент.

Таблицы *Exams, Credits, CourseWorks* содержат следующие столбцы: *id* – уникальный идентификатор соответствующего испытания; *Name* – название предмета по которому сдаётся испытание; *DateTest* – дата проведения испытания.

Таблицы *PassedExams, PassCredits, PassCourseWorks* содержат следующие столбцы: *id* – уникальный идентификатор сданного испытания; *CreditId, ExamId, CourseWorkId* – ссылка на сданный зачёт, экзамен или курсовую работу соответственно; *StudentId* – ссылка на студента, который сдал соответствующее испытание; *Mark* – оценка за курсовую работу или экзамен, *Mark* – оценивается значением 1 – сдал и значением 2 – не сдал.

Таблицы *Admins, TeacherLogins, StudentLogins* содержат следующие столбцы, которые являются общими для трёх таблиц: *id* – уникальный идентификатор; *Login* – имя пользователя, под которым он зарегистрирован в системе; *Password* – пароль для доступа к системе. Таблицы *Teachers* и *Admins* также содержат общие столбцы: *Name* – имя пользователя; *Surname* – фамилия пользователя; *Patronymic* – отчество пользователя. Таблица *Students* содержит столбец *StudentId*, который является ссылкой на студента; таблица *Teachers* содержит столбец *Position*, который указывает на должность преподавателя. Разработанная база данных называется *University* и состоит из четырнадцати взаимосвязанных таблиц. Все связи наглядно показывает диаграмма базы данных (Рисунок А.1).

2.3 Основные классы приложения

Для создания объектной модели, реализованы классы, описывающие различные сущности, относящиеся к предметной области задачи. Спроектированные классы реализованы по принципам *SOLID*. Все классы, описывающие такие сущности, реализованы в пространстве имён *Education* (Рисунок А.2).

Основными классами являются:

– *Faculty;*
– *Group;*
– *Specialty;*
– *Student;*
– *Teacher;*
– *CousreWork;*
– *Credit;*
– *Exam;*
– *UserAdmin;*
– *UserStudent;*
– *UserTeacher*.

Все классы имеют свойства, которые соответствуют столбцам из таблиц базы данных. Такое решение предоставляет простое создание и работу с объектами этих типов.

Класс *Test* является абстрактным классом и базовым для таких классов как *Exam, Credit, CourseWork*. Он содержит общие атрибуты для всех испытаний, такие как *DateTest* – дата проведения испытания; *Id* – уникальный идентификатор испытания; *Name* – название испытания; *GroupId* – идентификатор группы, студенты которой сдавали испытание в заданное время.

Соответственно каждый класс содержит такие же атрибуты, как и соответствующая таблица базы данных. Благодаря такой иерархии классов обеспечивается наиболее простой способ считывания, обработки и записи данных из объектов классов в соответствующие таблицы базы данных.

Как правило, все объекты классов хранятся в соответствующих коллекциях. Манипуляции с объектами классов реализуется при помощи технологии *LINQ*.

2.4 Структура *ORM* слоя

Для работы с данными из базы данных в *ORM* слое необходимо реализовать следующие основные операции: *Create* (добавление новой записи в таблицу), *Read* (чтение данных из таблицы или таблиц), *Update* (обновление данных в таблице), *Delete* (удаление данных из таблицы) (*CRUD*). Для реализации дан-

ного функционала необходимо создать интерфейс с декларированием вышеописанных методов, и в каждом классе, который будет работать с таблицами базы данных, реализовать данный интерфейс. Такое решение задачи представляет реализацию шаблона «Репозиторий».

Для доступа к данным из базы данных используется технология *ADO.NET*. При разработке приложения были использования основные классы данной технологии, первый работает для связи с СУБД *MySQL*, а именно *ConnectMySqlData* и для работы с БД: *ControllerCRUD, CourseWorkController, CreditController, ExamController, FacultyController* и *GroupController*.

Для взаимодействия с базой данных и реализации *ORM* слоя разработан проект *AccountingORM*, содержащий все необходимые классы для обработки базы данных.

Для классов, которые будут работать с таблицами пользователей необходимо создать интерфейс, отвечающий за создание пользователя по указанному логину и паролю.

IUserType – интерфейс, декларирующий *CRUD* операции для всех классов, реализующих его.

ListLogin – интерфейс, декларирующий метод получения пользователя по логину и паролю для всех классов, реализующих его.

Все классы, реализующие интерфейс *IUserType*, отвечают за обработку соответствующих таблиц в базе данных.

2.5 Структура многопользовательского доступа к системе

Приложение обеспечивает многопользовательский доступ к системе. За доступ к системе для разных видов пользователей разработаны классы, соответствующие таблицам из базы данных, которые хранят регистрационную информацию пользователей. Классы *StudentController* (для описания пользователя-студента), *TeacherController* (для описания пользователя-преподавателя) и *AdminController* (для описания пользователя-администратора) являются наследниками от абстрактного класса *User* (для описания пользователя), который имеет базовое описание любого пользователя – его логин и пароль. Классы *TeacherController* и *AdminController* расширяют класс *User*, добавляя такие атрибуты как фамилия, имя, отчество. Класс *AdminController* также содержит ссылку на соответствующий факультет. Класс *StudentController* содержит ссылку на соответствующего студента.

Данная иерархия позволяет реализовать многопользовательский доступ к системе. При входе в приложение, пользователь выбирает вид доступа и вводит данные для входа. Создается объект одного из приведенных выше типов и сопоставляется с уже существующими, которые выбираются из базы данных. Если созданный при входе в систему объект равен одному объекту среди созданных

при выборке из базы данных, значит пользователь ввел верные данные. В таком случае возвращается объект, являющийся описанием конкретного пользователя (администратора, студента или преподавателя).

С точки зрения проектирования, разработана диаграмма использования *UML*(*Unified Modeling Language*), представлена на рисунке 2.2.

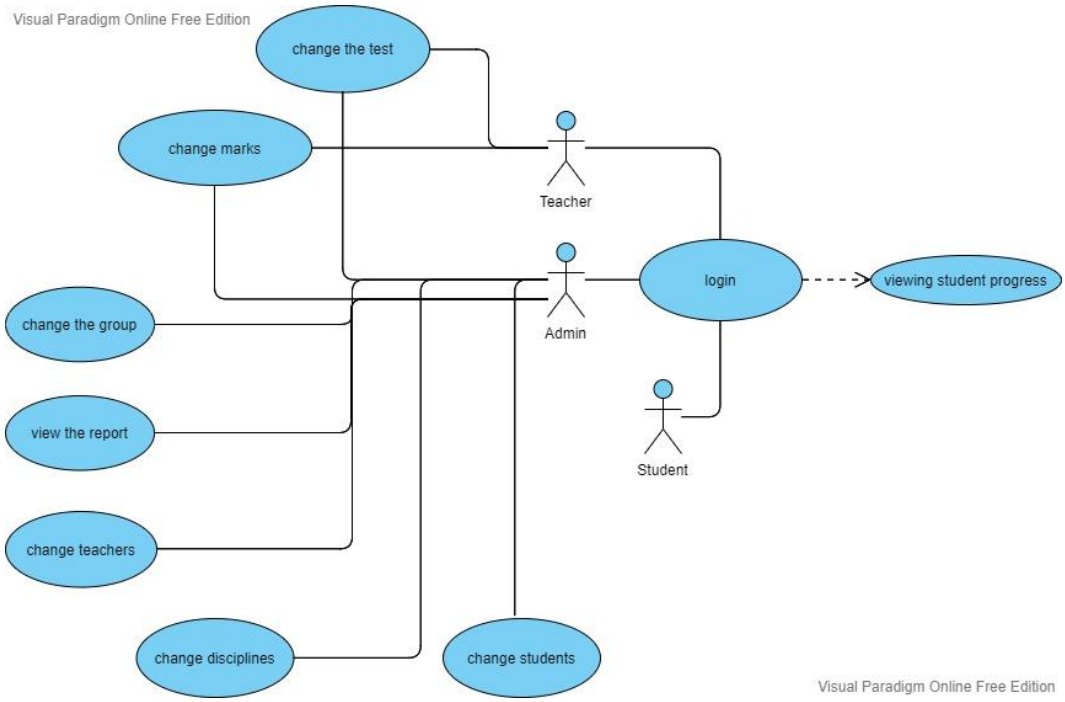

Рисунок 2.2 – Диаграмма использования *UML*(*Unified Modeling Language*)

Она состоит из графической диаграммы, описывающей действующие лица и прецеденты, а также спецификации, представляющего собой текстовое описание конкретных последовательностей действий (потока событий), которые выполняет пользователь при работе с системой.

3 ЭТАПЫ ПРОВЕДЕНИЯ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Графический интерфейс пользователя

Для создания графического интерфейса пользователя использовалась технология *Windows Forms*.

При создании окон приложения использованы основные элементы такие элементы интерфейса, такие как *Label, Button, TextBox, ComboBox, TabControl, ListView, MessageBox*.

Для отображения списков различным сущностей, использован элемент интерфейса *ListView*, представляющий таблицу. Пользователь взаимодействует с приложением посредством нажатия на кнопки (*Button*), выбор элементов из выпадающих списков (*ComboBox*), ввода текста в текстовые поля (*TextBox*).

Разработанное приложение является многооконным. В результате разработки было создано пять различных окон для различных целей:

– окно для ввода данных авторизации;

– окно для пользователя в роли студента;

– окно для пользователя в роли преподавателя и администратора;

– окно для ввода значений при изменении или добавлении данных;

– окно вывода списков необходимых данных.

В некоторых случаях открываются одни и те же окна с небольшими изменениями в отображении элементов управления. Это упрощает процесс разработки, уменьшает количество исходного кода приложения, уменьшает количество создаваемых форм и достигается при помощи повторного использования кода.

Разработанный графический интерфейс обеспечивает визуализацию всех возможностей взаимодействия с базой данных. Для разгрузки графического интерфейса на каждой форме создавалась небольшое количество элементов управления. Для простоты использования, поля ввода информации подписываются с указанием того, что в них должно быть внесено. Для исключения ошибок при выборе справочных критериев, используются выпадающие списки, где это возможно. Выпадающие списки предоставляют возможность выбора только тех критериев, которые есть в системе. Для текстовых полей ввода присутствует полная обработка ошибок и созданы проверки на корректность ввода данных.

3.2 Результаты проведения интегрированного тестирования

Проверка работоспособности приложения возможна при помощи использования самого приложения. Тестирование таким образом позволяет убедиться, что приложение полностью и правильно отображает данные, верно их изменяет и сохраняет. Такая проверка поможет выявить слабые места и возможные ошибки, которые очень сложно выявить модульным тестированием. Дополнительно, интегрированное тестирование поможет увидеть правильность отображения данных в графическом приложении.

Взаимодействия пользователя с приложением не должно вызывать ошибок или неполадок. Все непоследовательные действия пользователя должны быть предусмотрены и пользователю должно выводиться соответствующее сообщение о неверных действиях. Особое внимание уделено текстовым полям для ввода информации. Ввод некорректных данных проверяется и выводится соответствующее сообщение.

Для запуска программы необходимо открыть папку с проектом через *IDE Visual Studio*.

Окно входа в систему представлено на рисунках 3.1 – 3.3.

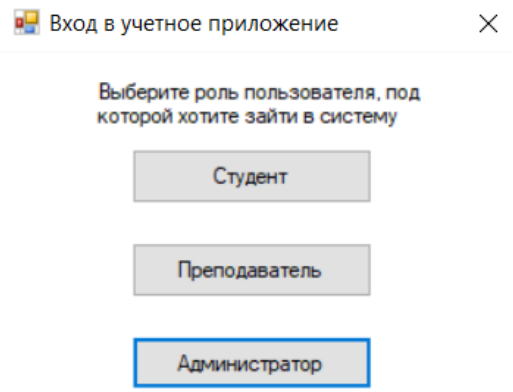

Рисунок 3.1 – Окно входа в систему

Будет открыто программное окно с выбором пользователя. При выборе одного из пользователя будет открыто окно с авторизацией пользователя. Вход в систему будет выполнен в случае, если введённые логин и пароль есть в базе данных. В противном случае на экране монитора будет выведено окно с сообщением об ошибке.

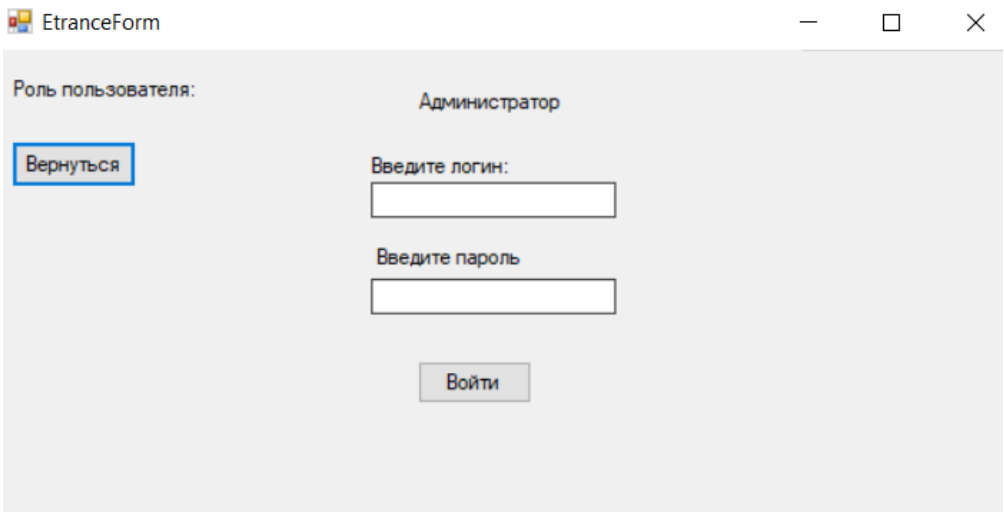

Рисунок 3.2 – Окно с авторизацией пользователя

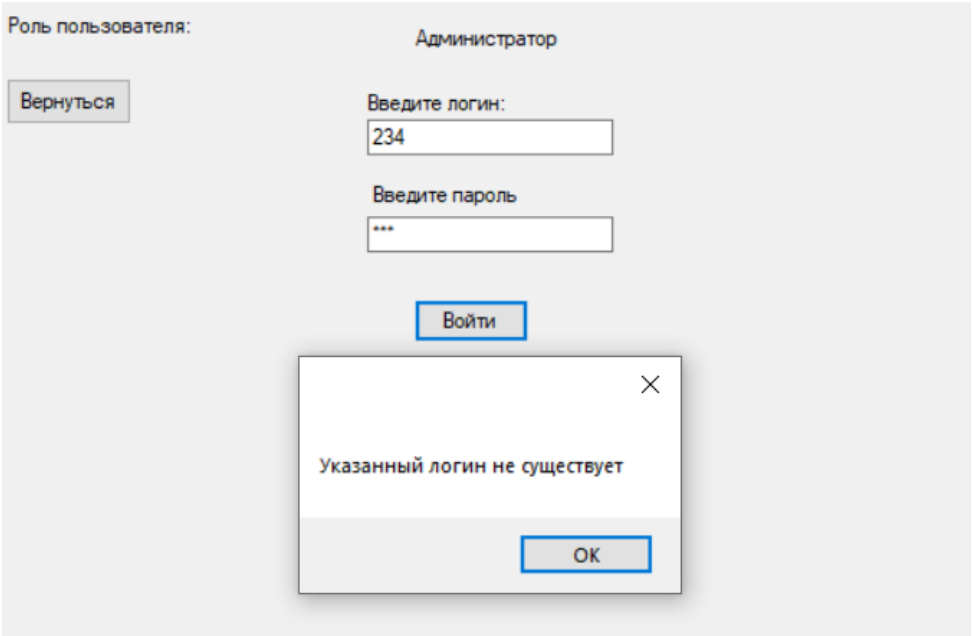

Рисунок 3.3 – Окно с сообщением об ошибке

Если закрыть окно, нажав на кнопку в правом верхнем углу, приложение закорет подключение к базе данных и завершит свою работу.

Для пользователя «Студент» доступны следующие функции:

– Просмотр всех сданных экзаменов;

– Просмотр всех сданных зачётов;

– Просмотр всех сданных курсовых работ.

Пример интерфейса пользователя «Студент» представлен на рисунке 3.4.

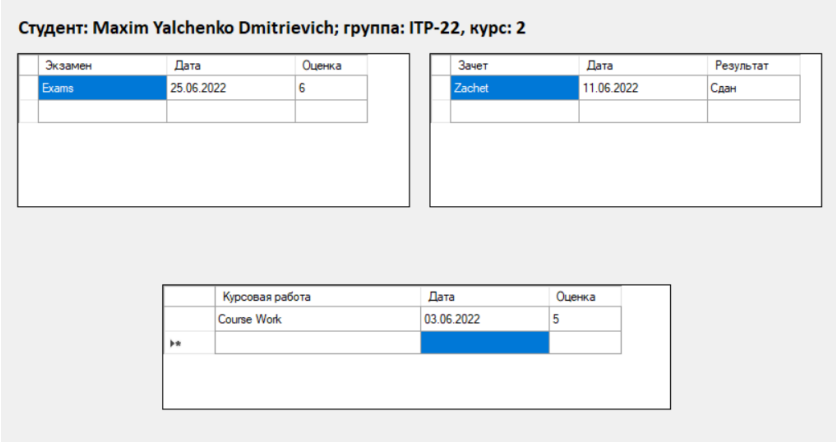

Рисунок 3.4 – Пример интерфейса пользователя «Студент»

Таким образом, программный продукт обеспечивает возможность пользователю «Студент» просматривать всю необходимую информацию только о себе, но без возможности её редактирования.

Для пользователя «Преподаватель» доступны следующие функции:

– Просмотр всех специальностей факультета;
– Просмотр всех групп факультета;
– Просмотр всех студентов выбранной группы;
– Просмотр всех сданных испытаний выбранного студента;
– Изменение оценки студента;
– Добавление и удаление испытания.

Пример интерфейса пользователя «Преподаватель» с выбранной вкладкой «Специальности» представлен на рисунке 3.5.

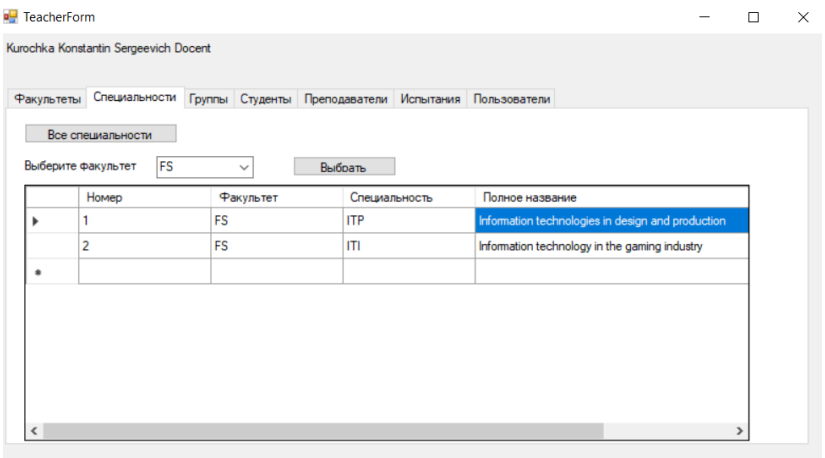

Рисунок 3.5 – Пример интерфейса пользователя «Преподаватель» с выбранной вкладкой «Специальности»

При выборе вкладки группы в окне будут отображены все группы, принадлежащие данному факультету.

Пример интерфейса пользователя «Преподаватель» с выбранной вкладкой «Группы» представлен на рисунке 3.9.

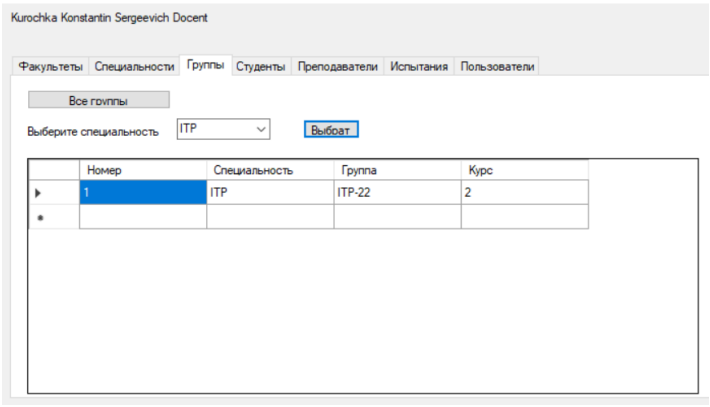

Рисунок 3. 9 – Пример интерфейса пользователя «Преподаватель» с выбранной вкладкой «Группы»

При нажатии на кнопку «Студенты» на экран монитора отобразится окно со списком всех студентов выбранной группы. Пользователь имеет возможность просмотреть успеваемость выбранного студента; изменить оценку за пройденное испытание. Окно с отображением всех студентов выбранной группы представлено на рисунке 3.10.

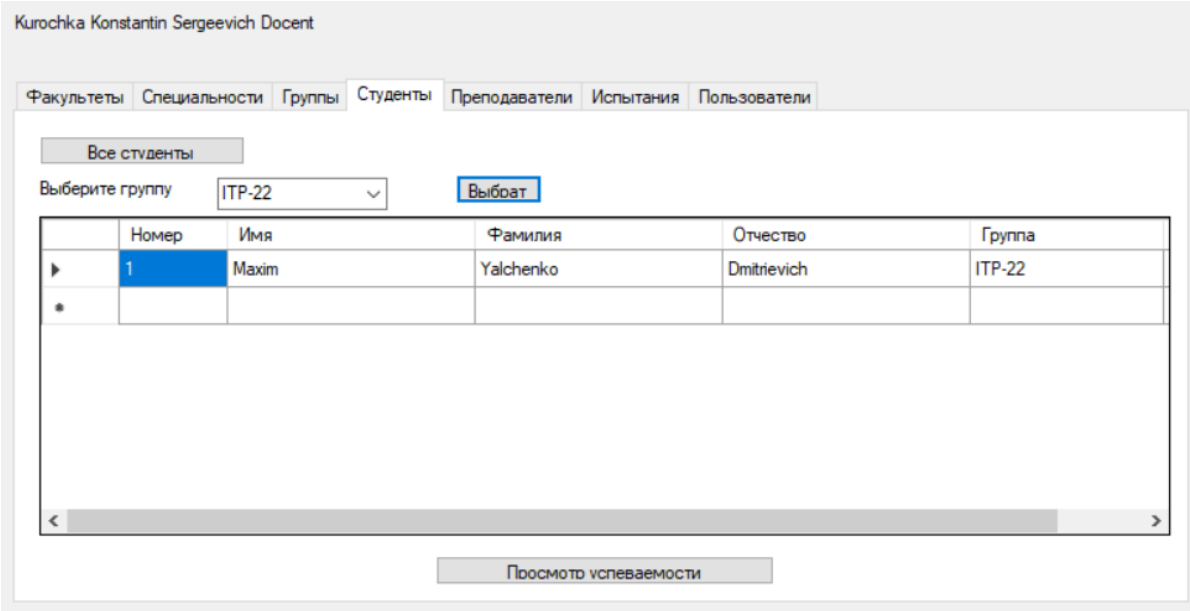

Рисунок 3.10 – Окно с отображением всех студентов выбранной группы

При нажатии на кнопку «Просмотр успеваемости» в окне «Студенты», программа откроет окно со всеми испытаниями для данного студента, в котором

преподаватель может изменить оценку. Окно отображения просмотра успеваемости представлено на рисунке 3.11.

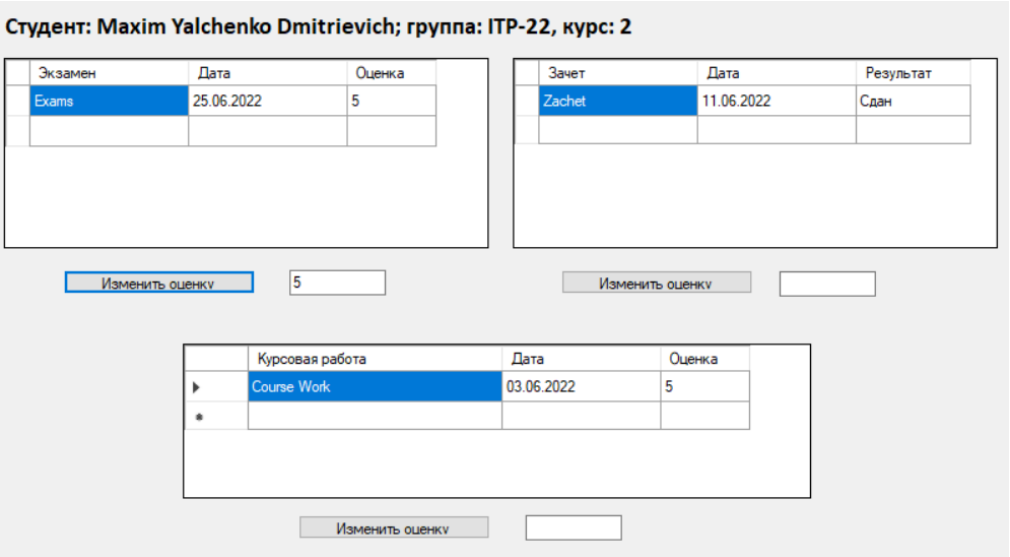

Рисунок 3.11 – Окно отображения просмотра успеваемости

Таким образом, программный продукт обеспечивает возможность пользователю «Преподаватель» просматривать всю информацию о студентах и изменять их оценки.

При нажатии на кнопку «Испытания», программа откроет окно со всеми испытаниями для выбранной группы, в котором преподаватель может добавлять, изменять или удалять испытания. Окно отображения испытаний представлено на рисунке 3.12.

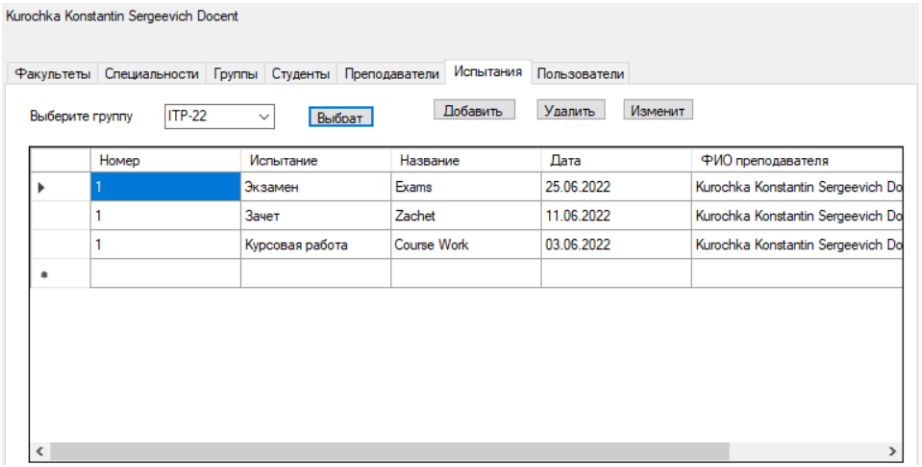

Рисунок 3.12 – Окно отображения просмотра испытаний

Окно редактирования испытания, представлено на рисунке 3.13.

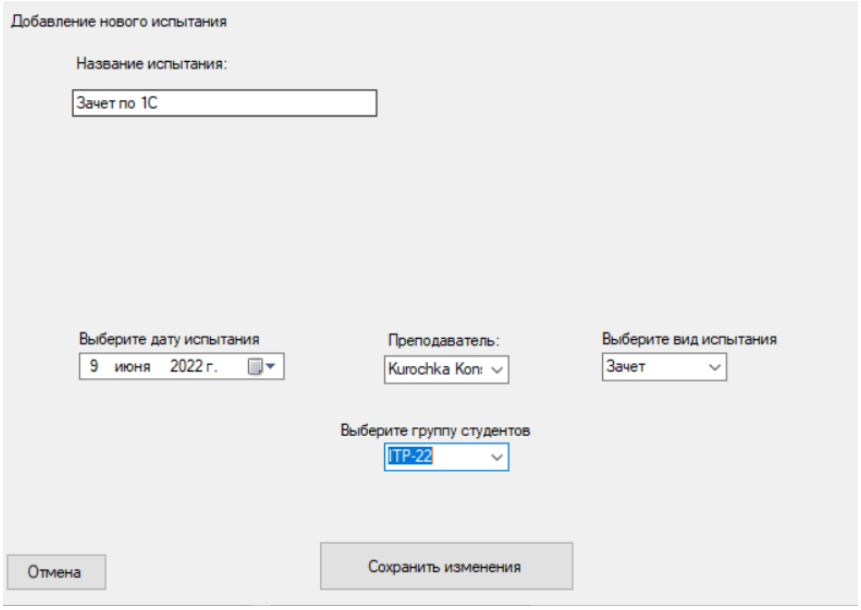

Рисунок 3.13 – Окно отображения редактирования испытаний

Для пользователя «Администратор» те же функции, что и для пользователя «Преподаватель» (глава 3 п. 3.3), но более расширенные. Он имеет доступ к просмотру и редактированию информации о каждом факультете, который есть в ВУЗе. Помимо этого, он может добавлять факультеты, удалять факультеты и регистрировать новых пользователей («Студент», «Преподаватель», «Администратор»). Пример графического интерфейса «Администратор» представлен на рисунке 3.14.

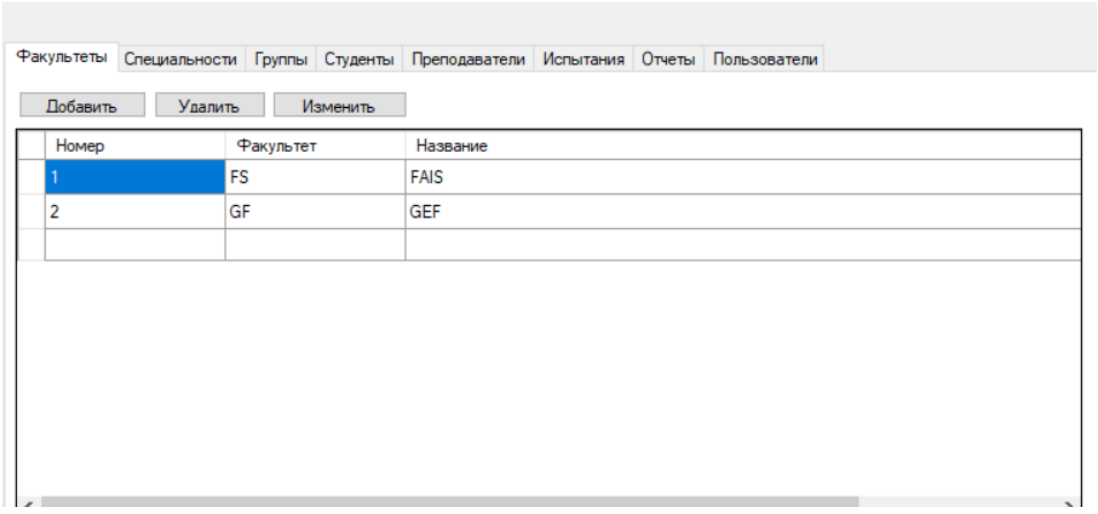

Рисунок 3.14 – Пример интерфейса «Администратор»

Особенной функцией администратора, является вывод информации в виде *XML* таблице о среднем балле по каждому студенту; отчёт о преподавателях; от-

чёт о лёгких и сложных испытаниях; отчёт об успевающих и не успевающих студентов. После формирования *XML* запроса администратору будет предложено его открыть в удобном формате через приложение *Exel*. Пример окна «Отчёт» в графическом интерфейсе «Администратор» представлен на рисунке 3.15.

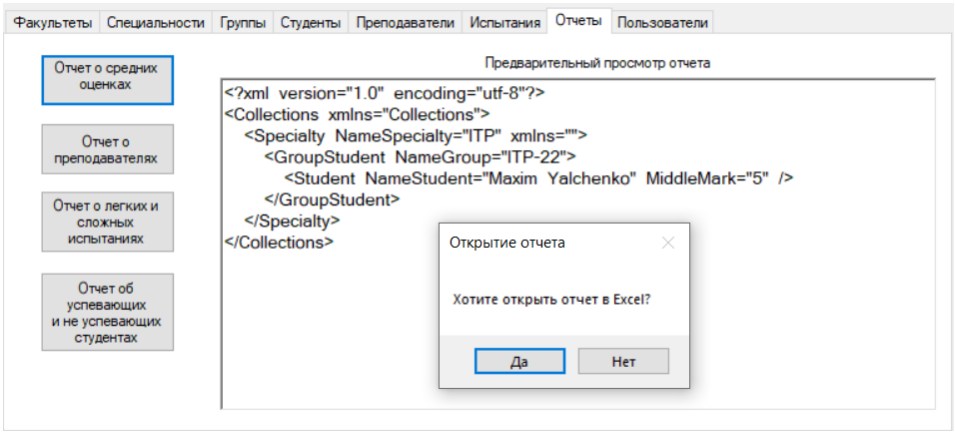

Рисунок 3.15 – Пример окна «Отчёт» в графическом интерфейсе «Администратор»

Пример окна изменения данных о преподавателе в графическом интерфейсе «Администратор» представлен на рисунке 3.16.

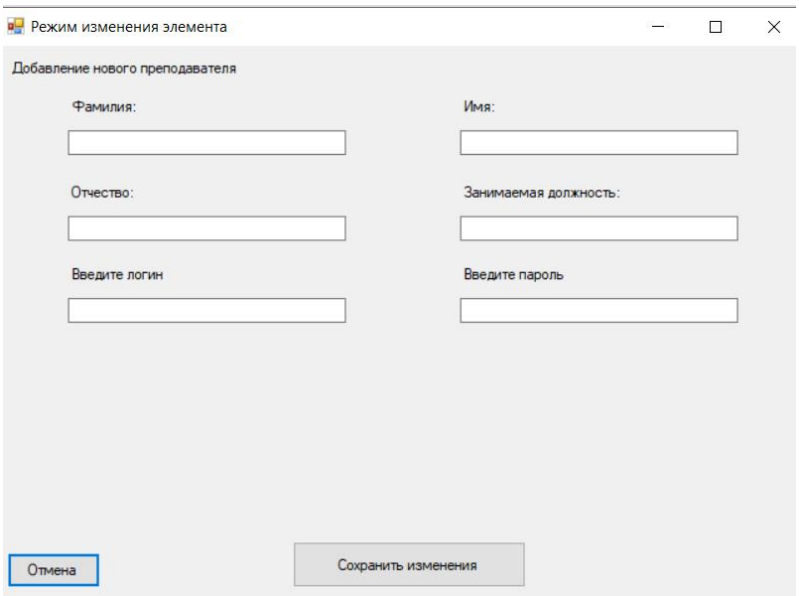

Рисунок 3.16 – Пример окна изменения данных о преподавателе в графическом интерфейсе «Администратор»

Таким образом, программный продукт обеспечивает пользователю «Администратор» полный доступ к системе, с возможностью её редактирования.

3.3 Результаты модульного тестирования

Модульные тесты необходимы для проверки работы приложения в различных условиях. Тесты проверяют реализованную бизнес-логику и покрывают большинство написанных методов. При написании методов нужно учитывать то, что после их написания, эти методы необходимо проверить.

Тестирование – это процесс исследования и испытания программного обеспечения, преследующий две задачи: убедиться в том, что программное обеспечение рабочее и соответствует требованиям, а также выявить ситуации, в которых поведение ПО является неправильным, нежелательным или не соответствует начальным требованиям.

Полноценные модульные тесты должны быть аналогичны работе реального приложения и придерживаться некоторых правил. При разработке модульных тестов необходимо придерживаться некоторых правил, согласно которых тесты должны:

– быть достоверными;

– быть независимыми;

– легко поддерживаться;

– быть понятными;

– соблюдать единую конвенцию именования.

Для написания модульных тестов, был разработан класс для тестирования объектов приложения.

Модульные тесты написаны на том же языке программирования, на котором написан весь исходный код. Для написания и проведения тестирования, создан проект модульного тестирования *AccountingTest*. Данный проект содержит два класса для тестирования двух проектов.

Класс *ItemUnitTest* проверяет методы классов из проекта *AccountingElement*. Данный класс имеет тестовые методы, проверяющие работу методов с коллекциями при помощи *LINQ to Object* и переопределенные методы *ToString*(), *Equals*(*object obj*), *GetHashCode*() во всем экземплярных классах. Данная проверка позволяет убедиться в правильности описания объектов и их представления в графическом интерфейсе пользователя.

3.4 Результаты верификации приложения

В результате проведения интегрированного и модульного тестирования разработанного приложения, можно убедиться, что приложение отвечает требованиям, характерным для системы, ведущей учет.

Тестирование показало, что приложение ведет себя стабильно, имеет обработку исключения и не допускает возникновение ошибок из-за действия пользователя.

Модульные тесты проверили работу бизнес-логики, взаимодействие с источником данных и методы, имеющие повышенную алгоритмическую сложность. Результат проведения модульных тестов представлен на рисунке 3.16.

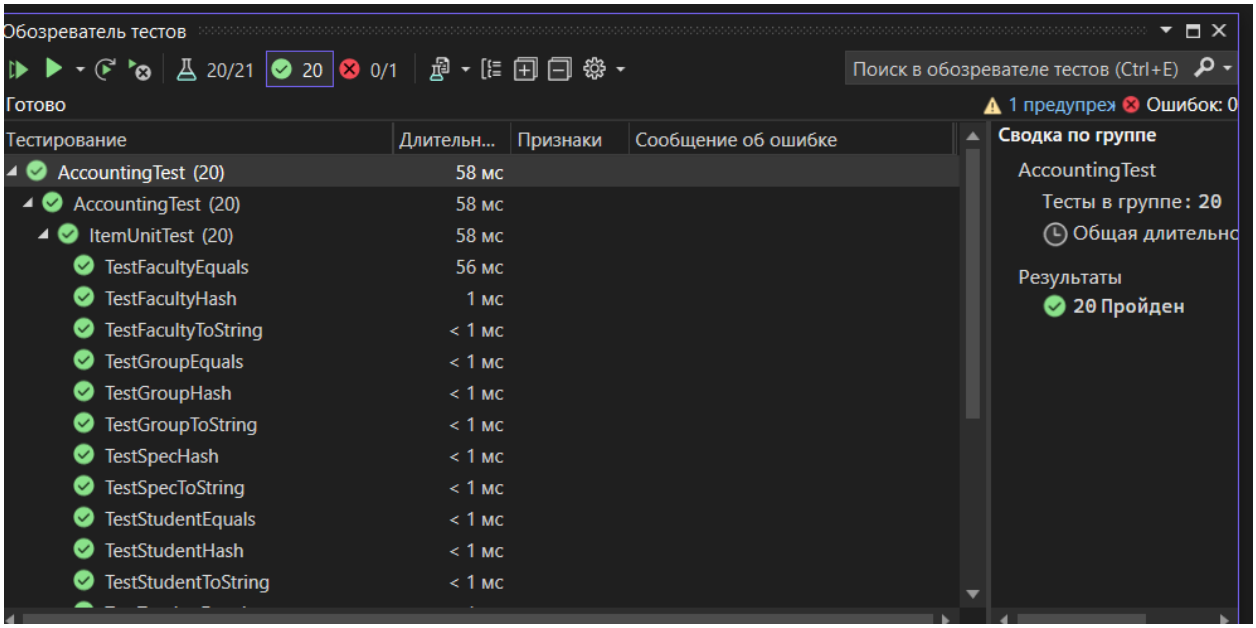

Рисунок 3.16 – Результат проведения модульных тестов

Проведение интегрированного тестирования показало работоспособность использования графического интерфейса, обработку исключений при нестандартных действиях пользователя. При этом интегрированное тестирование позволяет оценить общее состояние графического интерфейса, его простоту и логичность со стороны обычного пользователя.

Использование источника данных, отображение различной информации, возможность изменять, добавлять и удалять данные, многопользовательский доступ, формирование отчетов и наличие простого и функционального графического интерфейса предоставляет корректное ведение учета при помощи разработанного приложения. Успешно пройдены модульные тесты, что доказывает завершенность разработки приложения.

ЗАКЛЮЧЕНИЕ

Было разработано приложение для учёта успеваемости студентов во время обучения в ВУЗе. Приложение обеспечивает многопользовательский доступ к функциям системы (студент, преподаватель, администратор), позволяет создавать отчёты в формате *XML* файла по средним баллам студентов, находить самого лучшего и худшего студентов, анализировать и редактировать информацию.

Для написания продукта использовался язык программирования *C#*, платформа *.NET Framework*. В качестве источника данных использовалась СУБД *MySQL*. Для связи БД и программы использовалась технология *ADO.NET*. На её основе реализован *CRUD* для всех данных. Информация выводится на экран средствами технологии *Windows Forms*. Программный продукт может быть перенесён на другую технологию графического интерфейса или на другую базу данных с минимальными изменениями. Программа может расширяться и поддерживаться другими разработчиками по мере необходимости. Приложение протестировано модульными тестами, проведена опытная эксплуатация.

Достоинствами программы можно назвать:

– простой и понятный графический интерфейс;

– работа с «живыми» данными;

– многопользовательский доступ к системе, позволяющий скрыть или обеспечить определённый функционал для пользователя;

– возможность автоматического создания отчётов с дальнейшим открытием в электронных таблицах.

На основании всего вышеперечисленного следует, что приложение в полной мере решает задачу учёта успеваемости студентов за время обучения в ВУЗе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Джеффри Рихтер: Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. – Москва-Санкт-Петербург, 2012. – 22 с.

2. *Simon Kendal. Object Oriented Programming using C#*. – 1.1 часть.

3. *Sander Rossel. Object-Oriented Programming in C#* Succinctly. – 3 часть.

4. Карпова Т.С. Базы данных: модели, разработка, реализация. Учебник. СПб.: Питер, 2001.

5. Поль Дюбуа. *MySQL*: полное и исчерпывающее руководство по применению и администрированию.

6. Алан Бьюли. Изучаем *SQL*. Санкт-Петербург-Москва. – 20, 22 с.

7. Интегрированный языковой запрос (*LINQ*) [Электронный ресурс] –

Режим доступа *https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq*.

8. База данных [Электронный ресурс] – Режим доступа: *https://ru.wikipedia.org/wiki/База_данных*.

ПРИЛОЖЕНИЕ А
(обязательно)
Листинг программы

Faculty.cs

```
using System;

namespace AccountingElement
{
/// <summary>
/// Класс для описания факультета
/// </summary>
public class Faculty : IDealName
{
public int FacultId { get; set; }
public string ShortName { get; set; }
public string FullName { get; set; }

public Faculty(string shortName, string fullName)
{
ShortName = shortName;
FullName = fullName;
}

public Faculty(int facultId, string shortName, string fullName) : this(shortName, fullName)
{
FacultId = facultId;
}

public override string ToString() => $"{FacultId} {ShortName} {FullName}";

public override bool Equals(object obj)
{
if (obj == null)
return false;
Faculty faculty = obj as Faculty;
if (faculty == null)
return false;

if (faculty.FacultId != FacultId || faculty.ShortName != ShortName
|| faculty.FullName != FullName)
return false;
return true;
}

public override int GetHashCode()
{
return FacultId;
}
```

```
}
}
```

Group.cs

```
using System;

namespace AccountingElement
{
/// <summary>
/// Класс для описания группы студентов
/// </summary>
public class Group
{
public int GroupId { get; set; }
public string GroupName { get; set; }
public int Course { get; set; }
public int SpecId { get; set; }

public Group(string groupName, int course, int specId)
{
GroupName = groupName;
Course = course;
SpecId = specId;
}

public Group(int groupId, string groupName, int course, int specId)
: this(groupName, course, specId)
{
GroupId = groupId;
}

public override string ToString() => $"{GroupId} {GroupName} {Course} {SpecId}";

public override bool Equals(object obj)
{
if (obj == null)
return false;
Group group = obj as Group;
if (group == null)
return false;

if (group.GroupId != GroupId || group.GroupName != GroupName
|| group.SpecId != SpecId || group.Course != Course)
return false;
return true;
}

public override int GetHashCode()
{
return GroupId;
}

}
}
```

Speciality.cs

```
using System;

namespace AccountingElement
{
/// <summary>
/// Класс для описания специальности
/// </summary>
public class Specialty : IDealName
{
public int SpecId { get; set; }
public string ShortName { get; set; }
public string FullName { get; set; }
public int FacultId { get; set; }

public Specialty(string shortName, string fullName, int facultId)
{
ShortName = shortName;
FullName = fullName;
FacultId = facultId;
}

public Specialty(int specId, string shortName, string fullName, int facultId)
: this(shortName, fullName, facultId)
{
SpecId = specId;
}

public override string ToString() => $"{SpecId} {ShortName} {FullName} {FacultId}";

public override bool Equals(object obj)
{
if (obj == null)
return false;
Specialty spec = obj as Specialty;
if (spec == null)
return false;

if (spec.SpecId != SpecId || spec.ShortName != ShortName
|| spec.FullName != FullName || spec.FacultId != FacultId)
return false;
return true;
}

public override int GetHashCode()
{
return SpecId;
}

}
}
```

Student.cs

```
using System;
using AccountingElement.Tests;

namespace AccountingElement
{
/// <summary>
/// Класс для описания сущности студента
/// </summary>
public class Student : IPerson
{
public int StudentId { get; set; }
public string Name { get; set; }
public string Surname { get; set; }
public string Patronymic { get; set; }
public int GroupId { get; set; }

public Student(string name, string surname, string patronymic, int groupId)
{
Name = name;
Surname = surname;
Patronymic = patronymic;
GroupId = groupId;
}
public Student(int studentId, string name, string surname, string patronymic, int groupId)
: this(name, surname, patronymic, groupId)
{
StudentId = studentId;
}


public override string ToString() => $"{Surname} {Name} {Patronymic} {StudentId}";

public override bool Equals(object obj)
{
if (obj == null)
return false;

Student student = obj as Student;
if (student == null)
return false;

if (student.StudentId != StudentId || student.Name != Name || student.Surname != Surname
|| student.Patronymic != Patronymic || student.GroupId != GroupId)
return false;

return true;
}

public override int GetHashCode()
{
return StudentId;
}

}
}
```

Teacher.cs

```
using System;
using System.Data.Linq.Mapping;

namespace AccountingElement
{
/// <summary>
/// Класс для описания сущности преподавателя
/// </summary>
public class Teacher : IPerson
{
public int TeacherId { get; set; }
public string Name { get; set; }
public string Surname { get; set; }
public string Patronymic { get; set; }
public string Position { get; set; }
public string FullName => $"{Surname} {Name} {Patronymic} {Position}";

public Teacher(int teacherId, string name, string surname, string patronymic, string position)
: this(name, surname, patronymic, position)
{
TeacherId = teacherId;
}

public Teacher(string name, string surname, string patronymic, string position)
{
Name = name;
Surname = surname;
Patronymic = patronymic;
Position = position;
}

public override string ToString() =>
$"{Surname} {Name} {Patronymic} {Position}";

public override bool Equals(object obj)
{
if (obj == null)
return false;

Teacher teacher = obj as Teacher;
if (teacher == null)
return false;

if (teacher.TeacherId != TeacherId || teacher.Name != Name || teacher.Surname != Surname
|| teacher.Patronymic != Patronymic || teacher.Position != Position)
return false;

return true;
}

public override int GetHashCode()
{
return TeacherId;
```

```
}

}
}
```

AdminController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AccountingElement.UserAccess;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
public class AdminController : ControllerCRUD<User>
{
public override List<User> SelectItems()
{
List<User> admins = new List<User>();

MySqlCommand command = new MySqlCommand("select * from Admins") { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
User admin = new UserAdmin(reader.GetInt32("AdminId"), reader.GetString("Login"),
reader.GetString("Password"))
{ Name = reader.GetString("Name"), Surname = reader.GetString("Surname") };
admins.Add(admin);
}

reader.Close();
command.Dispose();
return admins;
}

public override void DeleteItem(User item)
{
MySqlCommand command = new MySqlCommand("delete from Admins " +
"where AdminId = (@Id)", connection);

command.Parameters.AddWithValue("@Id", item.Id);

command.ExecuteNonQuery();
command.Dispose();
}

public override void InsertItem(User item)
{
MySqlCommand command = new MySqlCommand("insert into Admins (Login, Password, Name, Surname)
" +
"values (@login, @password, @name, @surname)", connection);
```

```
command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@name", ((UserAdmin)item).Name);
command.Parameters.AddWithValue("@surname", ((UserAdmin)item).Surname);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(User item)
{
MySqlCommand command = new MySqlCommand("update Admins " +
"set Login = (@login), Password = (@password), Name = (@name), Surname = (@surname) " +
"where AdminId = (@Id)", connection);

command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@name", ((UserAdmin)item).Name);
command.Parameters.AddWithValue("@surname", ((UserAdmin)item).Surname);
command.Parameters.AddWithValue("@Id", item.Id);

command.ExecuteNonQuery();
command.Dispose();
}

}
}
```

ConectMySqlDatabase.cs

```
using System;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
   public class ConnectMySqlDatabase : IDisposable
   {
      private static ConnectMySqlDatabase Instance = null;
      private static readonly object ThreadLock = new object();

      private static string host = "localhost";
      private static int port = 3306;
      private static string database = "University";
      private static string username = "root";
      private static string password = "kikia2020";
      private static string stringConnection = $"Server={host};Database={database};port={port};" +
            $"User Id={username};password={password}";

      private readonly MySqlConnection Connection = null;

      private ConnectMySqlDatabase()
      {
         Connection = new MySqlConnection(stringConnection);
         Connection.Open();
      }
```

```
        public static ConnectMySqlDatabase GetInstance
        {
            get
            {
                lock (ThreadLock)
                {
                    if (Instance == null)
                        Instance = new ConnectMySqlDatabase();
                    return Instance;
                }
            }
        }

        public MySqlConnection GetConnection() => Connection;


        public void Dispose()
        {
            Connection.Close();
        }
    }
}
```

ControllerCRUD.cs

```
using System;
using System.Collections.Generic;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
    public abstract class ControllerCRUD<T> : IDisposable
    {
        private static ConnectMySqlDatabase instance = ConnectMySqlDatabase.GetInstance;
        protected MySqlConnection connection = instance.GetConnection();

        public abstract List<T> SelectItems();

        public abstract void InsertItem(T item);

        public abstract void DeleteItem(T item);

        public abstract void UpdateItem(T item);

        public virtual void DeleteItemById(int id)
        { }

        public void Dispose()
        {
            connection.Dispose();
            instance.Dispose();
        }

}
}
```

CourseWorkController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement.Tests;
using MySql.Data.MySqlClient;
using System.Linq;

namespace AccountingORM
{
public class CourseWorkController : ControllerCRUD<Test>, IConcreteTesting<Test>
{

public override List<Test> SelectItems()
{
List<Test> tests = new List<Test>();

string query = "select courseworks.CourseWorkId, Name, DateTest, TeacherId, GroupId, StudentId, Mark "
+
"from passcourseworks " +
"inner join courseworks on courseworks.CourseWorkId = PassCourseWorks.CourseWorkId";

MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
Test test = new CousreWork(reader.GetInt32("CourseWorkId"), reader.GetString("Name"), reader.Get-
DateTime("DateTest"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"))
{ Mark = (byte)reader.GetInt32("Mark"), StudentId = reader.GetInt32("StudentId") };
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public void InsertItemConcrete(int Mark, int TestId, int StudentId)
{
MySqlCommand command = new MySqlCommand("insert into PassCourseWorks (Mark, CourseWorkId,
StudentId) " +
"values (@mark, @courseWorkId, @studentId)", connection);

command.Parameters.AddWithValue("@mark", Mark);
command.Parameters.AddWithValue("@courseWorkId", TestId);
command.Parameters.AddWithValue("@studentId", StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public List<Test> SelectConcreate()
{
```

```
List<Test> tests = new List<Test>();
string query = "select CourseWorkId, Name, DateTest, TeacherId, GroupId from CourseWorks";

MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
Test test = new CousreWork(reader.GetInt32("CourseWorkId"), reader.GetString("Name"), reader.Get-
DateTime("DateTest"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"));
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public override void InsertItem(Test item)
{
MySqlCommand command = new MySqlCommand("insert into CourseWorks (Name, DateTest, TeacherId,
GroupId) " +
"values (@name, @dateTest, @teacherId, @groupId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
command.Dispose();

int id = SelectConcreate().Last().TestId;
StudentController studentController = new StudentController();
var list = studentController.SelectItems().Where(x => x.GroupId == item.GroupId).Select(x => x.Studen-
tId).ToList();
foreach (var i in list)
InsertItemConcrete(0, id, i);
}

public void InsertItemConcrete(Test item)
{
MySqlCommand command = new MySqlCommand("insert into PassCourseWorks (Mark, CourseWorkId,
StudentId) " +
"values (@mark, @courseworkId, @studentId)", connection);

command.Parameters.AddWithValue("@mark", item.Mark);
command.Parameters.AddWithValue("@courseworkId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Test item)
{
```

```
MySqlCommand command = new MySqlCommand("delete from CourseWorks " +
"where CourseWorkId = (@courseworkId)", connection);

command.Parameters.AddWithValue("@courseworkId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItemById(int id)
{
MySqlCommand command = new MySqlCommand("delete from CourseWorks " +
"where CourseWorkId = (@courseworkId)", connection);

command.Parameters.AddWithValue("@courseworkId", id);

command.ExecuteNonQuery();
command.Dispose();
}

public void DeleteItemConcrete(Test item)
{
MySqlCommand command = new MySqlCommand("delete from PassCourseWorks " +
"where CourseWorkId = (@courseworkId), StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@courseworkId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Test item)
{
MySqlCommand command = new MySqlCommand("update CourseWorks " +
"set Name = (@name), DateTest = (@dateTest), TeacherId = (@teacherId), GroupId = (@groupId) " +
"where CourseWorkId = (@courseworkId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);
command.Parameters.AddWithValue("@courseworkId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}


public void UpdateItemConcreate(Test item)
{

MySqlCommand command = new MySqlCommand("update PassCourseWorks " +
"set Mark = (@mark) " +
"where CourseWorkId = (@c" +
"ourseworkId) && StudentId = (@studentId)", connection);
```

```
command.Parameters.AddWithValue("@mark", item.Mark);
command.Parameters.AddWithValue("@courseworkId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();

}
}
}
```

CreditController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement.Tests;
using MySql.Data.MySqlClient;
using System.Linq;

namespace AccountingORM
{
public class CreditController : ControllerCRUD<Test>, IConcreteTesting<Test>
{

public override List<Test> SelectItems()
{
List<Test> tests = new List<Test>();
string query = "select Credits.CreditId, Name, DateTest, TeacherId, GroupId, StudentId, Mark " +
"from passCredit " +
"inner join Credits on Credits.CreditId = passCredit.CreditId";

MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
Test test = new Credit(reader.GetInt32("CreditId"), reader.GetString("Name"), reader.GetDateTime("Date-
Test"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"))
{ Mark = (byte)reader.GetInt32("Mark"), StudentId = reader.GetInt32("StudentId") };
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public List<Test> SelectConcreate()
{
List<Test> tests = new List<Test>();
string query = "select CreditId, Name, DateTest, TeacherId, GroupId from Credits";

MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();
```

```
while (reader.Read())
{
Test test = new Credit(reader.GetInt32("CreditId"), reader.GetString("Name"), reader.GetDateTime("Date-
Test"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"));
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public void InsertItemConcrete(int Mark, int TestId, int StudentId)
{
MySqlCommand command = new MySqlCommand("insert into PassCredit (Mark, CreditId, StudentId) " +
"values (@mark, @creditId, @studentId)", connection);

command.Parameters.AddWithValue("@mark", Mark);
command.Parameters.AddWithValue("@creditId", TestId);
command.Parameters.AddWithValue("@studentId", StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void InsertItem(Test item)
{
MySqlCommand command = new MySqlCommand("insert into Credits (Name, DateTest, TeacherId,
GroupId) " +
"values (@name, @dateTest, @teacherId, @groupId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
command.Dispose();

int id = SelectConcreate().Last().TestId;
StudentController studentController = new StudentController();
var list = studentController.SelectItems().Where(x => x.GroupId == item.GroupId).Select(x => x.Studen-
tId).ToList();
foreach (var i in list)
InsertItemConcrete(0, id, i);
}

public void InsertItemConcrete(Test item)
{
MySqlCommand command = new MySqlCommand("insert into PassCredits (Mark, CreditId, StudentId) " +
"values (@mark, @creditId, @studentId)", connection);

command.Parameters.AddWithValue("@mark", item.Mark);
command.Parameters.AddWithValue("@creditId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);
```

```
command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Test item)
{
MySqlCommand command = new MySqlCommand("delete from Credits " +
"where CreditId = (@creditId)", connection);

command.Parameters.AddWithValue("@creditId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItemById(int id)
{
MySqlCommand command = new MySqlCommand("delete from Credits " +
"where CreditId = (@creditId)", connection);

command.Parameters.AddWithValue("@creditId", id);

command.Dispose();
}

public void DeleteItemConcrete(Test item)
{
MySqlCommand command = new MySqlCommand("delete from PassCredits " +
"where CreditId = (@creditId), StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@creditId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Test item)
{
MySqlCommand command = new MySqlCommand("update Credits " +
"set Name = (@name), DateTest = (@dateTest), TeacherId = (@teacherId), GroupId = (@groupId) " +
"where CreditId = (@creditId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);
command.Parameters.AddWithValue("@creditId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}


public void UpdateItemConcreate(Test item)
{
MySqlCommand command = new MySqlCommand("update PassCredit " +
```

```
"set Mark = (@mark)" +
"where CreditId = (@creditId) && StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@mark", item.Mark);
command.Parameters.AddWithValue("@creditId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

ExamController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement.Tests;
using MySql.Data.MySqlClient;
using System.Linq;
using System.Text;
namespace AccountingORM
{

public class ExamController : ControllerCRUD<Test>, IConcreteTesting<Test>
{


public override List<Test> SelectItems()
{
List<Test> tests = new List<Test>();
string query = "select exams.ExamId, Name, DateTest, TeacherId, GroupId, StudentId, Mark " +
"from passexams " +
"inner join exams on exams.ExamId = passexams.ExamId";

MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
Test test = new Exam(reader.GetInt32("ExamId"), reader.GetString("Name"), reader.GetDateTime("Date-
Test"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"))
{ Mark = (byte)reader.GetInt32("Mark"), StudentId = reader.GetInt32("StudentId") };
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public List<Test> SelectConcreate()
{
List<Test> tests = new List<Test>();
string query = "select ExamId, Name, DateTest, TeacherId, GroupId from Exams";
```

```
MySqlCommand command = new MySqlCommand(query) { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
{
Test test = new Exam(reader.GetInt32("ExamId"), reader.GetString("Name"), reader.GetDateTime("Date-
Test"),
reader.GetInt32("TeacherId"), reader.GetInt32("GroupId"));
tests.Add(test);
}

reader.Close();
command.Dispose();
return tests;
}

public override void InsertItem(Test item)
{
MySqlCommand command = new MySqlCommand("insert into Exams (Name, DateTest, TeacherId,
GroupId) " +
"values (@name, @dateTest, @teacherId, @groupId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
command.Dispose();

int id = SelectConcreate().Last().TestId;
StudentController studentController = new StudentController();
var list = studentController.SelectItems().Where(x => x.GroupId == item.GroupId).Select(x => x.Studen-
tId).ToList();
foreach (var i in list)
InsertItemConcrete(0, id, i);
}

public void InsertItemConcrete(int Mark, int TestId, int StudentId)
{
MySqlCommand command = new MySqlCommand("insert into PassExams (Mark, ExamId, StudentId) " +
"values (@mark, @examId, @studentId)", connection);

command.Parameters.AddWithValue("@mark", Mark);
command.Parameters.AddWithValue("@examId", TestId);
command.Parameters.AddWithValue("@studentId", StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Test item)
{
MySqlCommand command = new MySqlCommand("delete from Exams " +
"where ExamId = (@examId)", connection);
```

```
command.Parameters.AddWithValue("@examId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItemById(int id)
{
MySqlCommand command = new MySqlCommand("delete from Exams " +
"where ExamId = (@examId)", connection);

command.Parameters.AddWithValue("@examId", id);

command.ExecuteNonQuery();
command.Dispose();
}

public void DeleteItemConcrete(Test item)
{
MySqlCommand command = new MySqlCommand("delete from PassExams " +
"where ExamId = (@examId) and StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@examId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Test item)
{
MySqlCommand command = new MySqlCommand("update Exams " +
"set Name = (@name), DateTest = (@dateTest), TeacherId = (@teacherId), GroupId = (@groupId) " +
"where ExamId = (@examId)", connection);

command.Parameters.AddWithValue("@name", item.NameTest);
command.Parameters.AddWithValue("@dateTest", item.DateTest);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);
command.Parameters.AddWithValue("@groupId", item.GroupId);
command.Parameters.AddWithValue("@examId", item.TestId);

command.ExecuteNonQuery();
command.Dispose();
}

public void UpdateItemConcreate(Test item)
{

MySqlCommand command = new MySqlCommand("update PassExams " +
"set Mark = (@mark) " +
"where ExamId = (@examId) && StudentId = (@studentId)", connection);
byte[] byteArrayEncoding = Encoding.UTF8.GetBytes(command.CommandText);
command.CommandText = Encoding.Default.GetString(byteArrayEncoding);
command.Parameters.AddWithValue("@mark", item.Mark);
command.Parameters.AddWithValue("@examId", item.TestId);
command.Parameters.AddWithValue("@studentId", item.StudentId);
command.ExecuteNonQuery();
```

```
command.Dispose();
}
}

}
```

FacultyController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
public class FacultyController : ControllerCRUD<Faculty>
{

public override List<Faculty> SelectItems()
{
List<Faculty> faculties = new List<Faculty>();

MySqlCommand command = new MySqlCommand("select * from Faculties") { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
faculties.Add(new Faculty(reader.GetInt32("FacultId"), reader.GetString("ShortName"),
reader.GetString("FullName")));

reader.Close();
command.Dispose();
return faculties;
}

public override void InsertItem(Faculty item)
{
MySqlCommand command = new MySqlCommand("insert into Faculties (ShortName, FullName) " +
"values (@shortName, @fullName)", connection);

command.Parameters.AddWithValue("@shortName", item.ShortName);
command.Parameters.AddWithValue("@fullName", item.FullName);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Faculty item)
{
MySqlCommand command = new MySqlCommand("delete from Faculties " +
"where FacultId = (@facultId)", connection);

command.Parameters.AddWithValue("@facultId", item.FacultId);

command.ExecuteNonQuery();
command.Dispose();
}
```

```
public override void DeleteItemById(int id)
{
MySqlCommand command = new MySqlCommand("delete from Faculties " +
"where FacultId = (@facultId)", connection);

command.Parameters.AddWithValue("@facultId", id);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Faculty item)
{
MySqlCommand command = new MySqlCommand("update Faculties " +
"set ShortName = (@shortName), FullName = (@fullName) where FacultId = (@facultId)", connection);

command.Parameters.AddWithValue("@shortName", item.ShortName);
command.Parameters.AddWithValue("@fullName", item.FullName);
command.Parameters.AddWithValue("@facultId", item.FacultId);

command.ExecuteNonQuery();
command.Dispose();



}

}
}
```

GroupController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
public class GroupController : ControllerCRUD<Group>
{
public override List<Group> SelectItems()
{
List<Group> groups = new List<Group>();

MySqlCommand command = new MySqlCommand("select * from Studentgroups") { Connection = connec-
tion };
var reader = command.ExecuteReader();

while (reader.Read())
groups.Add(new Group(
reader.GetInt32("GroupId"), reader.GetString("GroupName"),
reader.GetInt32("Course"), reader.GetInt32("SpecId")));

reader.Close();
```

```
command.Dispose();
return groups;
}

public override void InsertItem(Group item)
{
MySqlCommand command = new MySqlCommand("insert into Studentgroups (GroupName, Course, Spe-
cId) " +
"values (@groupName, @course, @specId)", connection);

command.Parameters.AddWithValue("@groupName", item.GroupName);
command.Parameters.AddWithValue("@course", item.Course);
command.Parameters.AddWithValue("@specId", item.SpecId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Group item)
{
MySqlCommand command = new MySqlCommand("delete from Studentgroups " +
"where GroupId = (@groupId)", connection);

command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Group item)
{
MySqlCommand command = new MySqlCommand("update Studentgroups " +
"set GroupName = (@groupName), Course = (@course), SpecId = (@specId) " +
"where GroupId = (@groupId)", connection);

command.Parameters.AddWithValue("@groupName", item.GroupName);
command.Parameters.AddWithValue("@course", item.Course);
command.Parameters.AddWithValue("@specId", item.SpecId);
command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

SpecialityController.cs

```
using System;
using System.Collections.Generic;
using AccountingElement;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
public class SpecialtyController : ControllerCRUD<Specialty>
```

```
{
public override List<Specialty> SelectItems()
{
List<Specialty> specialties = new List<Specialty>();

MySqlCommand command = new MySqlCommand("select * from Specialties") { Connection = connection
};
var reader = command.ExecuteReader();

while (reader.Read())
specialties.Add(new Specialty(
reader.GetInt32("SpecId"), reader.GetString("ShortName"),
reader.GetString("FullName"), reader.GetInt32("FacultId")));

reader.Close();
command.Dispose();
return specialties;
}

public override void InsertItem(Specialty item)
{
MySqlCommand command = new MySqlCommand("insert into Specialties (ShortName, FullName, Facul-
tId) " +
"values (@shortName, @fullName, @facultId)", connection);

command.Parameters.AddWithValue("@shortName", item.ShortName);
command.Parameters.AddWithValue("@fullName", item.FullName);
command.Parameters.AddWithValue("@facultId", item.FacultId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Specialty item)
{
MySqlCommand command = new MySqlCommand("delete from Specialties " +
"where SpecId = (@specId)", connection);

command.Parameters.AddWithValue("@specId", item.SpecId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItemById(int id)
{
MySqlCommand command = new MySqlCommand("delete from Specialties " +
"where SpecId = (@specId)", connection);

command.Parameters.AddWithValue("@specId", id);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Specialty item)
{
```

```
MySqlCommand command = new MySqlCommand("update Specialties " +
"set ShortName = (@shortName), FullName = (@fullName), FacultId = (@facultId) " +
"where SpecId = (@specId)", connection);

command.Parameters.AddWithValue("@shortName", item.ShortName);
command.Parameters.AddWithValue("@fullName", item.FullName);
command.Parameters.AddWithValue("@facultId", item.FacultId);
command.Parameters.AddWithValue("@specId", item.SpecId);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

StudentController.cs

```
using System;
using System.Collections.Generic;
using MySql.Data.MySqlClient;
using AccountingElement;
using System.Linq;

namespace AccountingORM
{
public class StudentController : ControllerCRUD<Student>
{
public override List<Student> SelectItems()
{
List<Student> students = new List<Student>();

MySqlCommand command = new MySqlCommand("select * from Students") { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
students.Add(new Student(
reader.GetInt32("StudentId"), reader.GetString("Name"), reader.GetString("Surname"),
reader.GetString("Patronymic"), reader.GetInt32("GroupId")));

reader.Close();
command.Dispose();
return students;
}

public override void InsertItem(Student item)
{
MySqlCommand command = new MySqlCommand("insert into Students (Name, Surname, Patronymic,
GroupId) " +
"values (@name, @surname, @patronymic, @groupId)", connection);

command.Parameters.AddWithValue("@name", item.Name);
command.Parameters.AddWithValue("@surname", item.Surname);
command.Parameters.AddWithValue("@patronymic", item.Patronymic);
command.Parameters.AddWithValue("@groupId", item.GroupId);

command.ExecuteNonQuery();
```

```
command.Dispose();

int studentId = SelectItems().Last().StudentId;
ExamController examController = new ExamController();
CourseWorkController courseWorkController = new CourseWorkController();
CreditController creditController = new CreditController();
var test = examController.SelectConcreate().Select(e => e).Where(i => i.GroupId == item.GroupId);
foreach (var i in test.ToList().Distinct())
examController.InsertItemConcrete(0, i.TestId, studentId);
test = courseWorkController.SelectConcreate().Select(e => e).Where(i => i.GroupId == item.GroupId);
foreach (var i in test.ToList().Distinct())
courseWorkController.InsertItemConcrete(0, i.TestId, studentId);
test = creditController.SelectConcreate().Select(e => e).Where(i => i.GroupId == item.GroupId);
foreach (var i in test.ToList().Distinct())
creditController.InsertItemConcrete(0, i.TestId, studentId);
}

public override void DeleteItem(Student item)
{
MySqlCommand command = new MySqlCommand("delete from Students " +
"where StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Student item)
{
MySqlCommand command = new MySqlCommand("update Students " +
"set Name = (@name), Surname = (@surname), Patronymic = (@patr), GroupId = (@groupId) " +
"where StudentId = (@studentId)", connection);

command.Parameters.AddWithValue("@name", item.Name);
command.Parameters.AddWithValue("@surname", item.Surname);
command.Parameters.AddWithValue("@patr", item.Patronymic);
command.Parameters.AddWithValue("@groupId", item.GroupId);
command.Parameters.AddWithValue("@studentId", item.StudentId);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

StudentLoginController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;
using AccountingElement.UserAccess;
```

```
namespace AccountingORM
{
public class StudentLoginController : ControllerCRUD<User>
{
public override List<User> SelectItems()
{
List<User> students = new List<User>();

MySqlCommand command = new MySqlCommand("select * from StudentLogins") { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
students.Add(new UserStudent(
Convert.ToInt32(reader["StudentLoginId"]), reader["Login"].ToString(),
reader["Password"].ToString(), Convert.ToInt32(reader["StudentId"])));

reader.Close();
command.Dispose();
return students;
}

public override void DeleteItem(User item)
{
MySqlCommand command = new MySqlCommand("delete from StudentLogins " +
"where StudentLoginId = (@Id)", connection);

command.Parameters.AddWithValue("@Id", item.Id);

command.ExecuteNonQuery();
command.Dispose();
}

public override void InsertItem(User item)
{
MySqlCommand command = new MySqlCommand("insert into StudentLogins (Login, Password, StudentId) " +
"values (@login, @password, @studentId)", connection);

command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@studentId", item.UserId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(User item)
{
MySqlCommand command = new MySqlCommand("update StudentLogins " +
"set Login = (@login), Password = (@password) " +
"where StudentLoginId = (@id)", connection);

command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@id", item.Id);
```

```
command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

TeacherController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AccountingElement;
using MySql.Data.MySqlClient;

namespace AccountingORM
{
public class TeacherController : ControllerCRUD<Teacher>
{
public override List<Teacher> SelectItems()
{
List<Teacher> teachers = new List<Teacher>();

MySqlCommand command = new MySqlCommand("select * from Teachers") { Connection = connection };
var reader = command.ExecuteReader();

while (reader.Read())
teachers.Add(new Teacher(
reader.GetInt32("TeacherId"), reader.GetString("Name"), reader.GetString("Surname"),
reader.GetString("Patronymic"), reader.GetString("Position")));

reader.Close();
command.Dispose();
return teachers;
}

public override void InsertItem(Teacher item)
{
MySqlCommand command = new MySqlCommand("insert into Teachers (Name, Surname, Patronymic, Po-
sition) " +
"values (@name, @surname, @patronymic, @position)", connection);

command.Parameters.AddWithValue("@name", item.Name);
command.Parameters.AddWithValue("@surname", item.Surname);
command.Parameters.AddWithValue("@patronymic", item.Patronymic);
command.Parameters.AddWithValue("@position", item.Position);

command.ExecuteNonQuery();
command.Dispose();
}

public override void DeleteItem(Teacher item)
{
MySqlCommand command = new MySqlCommand("delete from Teachers " +
"where TeacherId = (@teacherId)", connection);
```

```
command.Parameters.AddWithValue("@teacherId", item.TeacherId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(Teacher item)
{
MySqlCommand command = new MySqlCommand("update Teachers " +
"set Name = (@name), Surname = (@surname), Patronymic = (@patr), Position = (@position) " +
"where TeacherId = (@teacherId)", connection);

command.Parameters.AddWithValue("@name", item.Name);
command.Parameters.AddWithValue("@surname", item.Surname);
command.Parameters.AddWithValue("@patr", item.Patronymic);
command.Parameters.AddWithValue("@position", item.Position);
command.Parameters.AddWithValue("@teacherId", item.TeacherId);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

TeacherLoginController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MySql.Data.MySqlClient;
using AccountingElement.UserAccess;

namespace AccountingORM
{
public class TeacherLoginController : ControllerCRUD<User>
{
public override List<User> SelectItems()
{
List<User> students = new List<User>();

MySqlCommand command = new MySqlCommand("select * from TeacherLogins") { Connection = connec-
tion };
var reader = command.ExecuteReader();

while (reader.Read())
students.Add(new UserTeacher(
Convert.ToInt32(reader["TeacherLoginId"]), reader["Login"].ToString(),
reader["Password"].ToString(), Convert.ToInt32(reader["TeacherId"])));

reader.Close();
command.Dispose();
return students;
}
```

```
public override void DeleteItem(User item)
{
MySqlCommand command = new MySqlCommand("delete from TeacherLogins " +
"where TeacherLoginId = (@Id)", connection);

command.Parameters.AddWithValue("@Id", item.Id);

command.ExecuteNonQuery();
command.Dispose();
}

public override void InsertItem(User item)
{
MySqlCommand command = new MySqlCommand("insert into TeacherLogins (Login, Password,
TeacherId) " +
"values (@login, @password, @teacherId)", connection);

command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@teacherId", item.UserId);

command.ExecuteNonQuery();
command.Dispose();
}

public override void UpdateItem(User item)
{
MySqlCommand command = new MySqlCommand("update TeacherLogins " +
"set Login = (@login), Password = (@password) " +
"where TeacherLoginId = (@id)", connection);

command.Parameters.AddWithValue("@login", item.Login);
command.Parameters.AddWithValue("@password", item.Password);
command.Parameters.AddWithValue("@id", item.Id);

command.ExecuteNonQuery();
command.Dispose();
}
}
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AccountingUI
{
static class Program
{
/// <summary>
/// Главная точка входа для приложения.
/// </summary>
```

```
[STAThread]
static void Main()
{
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new SelectRole());
}
}
}
```

ItemUnitTest.cs

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using AccountingElement;
using AccountingElement.Tests;
using AccountingElement.UserAccess;

namespace AccountingTest
{
/// <summary>
/// Класс для тестирования объектов приложения
/// </summary>
[TestClass]
public class ItemUnitTest
{
Faculty faculty = new Faculty(1, "FS", "FAIS");
Specialty spec = new Specialty(1, "ITP", "Information systems and technologies in design and production", 1);
Group group = new Group(1, "ITP-22", 2, 1);
Student student = new Student(1, "Yalchenko", "Maxim", "Dmitrievich", 1);
Teacher teacher = new Teacher(1, "Petr", "Sergeev", "Grigorevich", "Docent");
User user = new UserStudent(1, "111", "222");
Test test = new Exam(13, "Math exam", new DateTime(2022, 06, 16), 1, 1);

[TestMethod]
public void TestFacultyToString()
{
var except = "1 FS FAIS";
var actual = faculty.ToString();
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestFacultyEquals()
{
var f2 = new Faculty(100, "FS", "FAIS");
Assert.IsFalse(faculty.Equals(f2));
f2.FacultId = 1;
Assert.IsTrue(faculty.Equals(f2));
}
[TestMethod]
public void TestFacultyHash()
{
Assert.AreEqual(faculty.GetHashCode(), 1);
}

[TestMethod]
public void TestSpecToString()
{
var except = "1 ITP Information systems and technologies in design and production 1";
var actual = spec.ToString();
```

```
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestSpecEquals()
{
Specialty s2 = new Specialty(1, "ITP", "Information systems and technologies in design and production", 1);
Assert.IsFalse(spec.Equals(s2));
s2.FacultId = 3;
Assert.IsTrue(spec.Equals(s2));
}
[TestMethod]
public void TestSpecHash()
{
Assert.AreEqual(spec.GetHashCode(), 1);
}


[TestMethod]
public void TestGroupToString()
{
var except = "1 ITP-22 2 1";
var actual = group.ToString();
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestGroupEquals()
{
Assert.IsFalse(group.Equals(null));
}
[TestMethod]
public void TestGroupHash()
{
Assert.AreEqual(group.GetHashCode(), 1);
}


[TestMethod]
public void TestStudentToString()
{
var except = "Maxim Yalchenko Dmitrievich 1";
var actual = student.ToString();
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestStudentEquals()
{
Assert.IsTrue(student.Equals(student));
}
[TestMethod]
public void TestStudentHash()
{
Assert.AreEqual(student.GetHashCode(), 1);
}


[TestMethod]
public void TestTeacherToString()
{
var except = "Sergeev Petr Grigorevich Docent";
var actual = teacher.ToString();
Assert.AreEqual(except, actual);
}
```

```
[TestMethod]
public void TestTeacherEquals()
{
Assert.IsTrue(teacher.Equals(new Teacher(1, "Petr", "Sergeev", "Grigorevich", "Docent")));
}
[TestMethod]
public void TestTeacherHash()
{
Assert.AreEqual(teacher.GetHashCode(), 1);
}


[TestMethod]
public void TestUserToString()
{
var except = "111";
var actual = user.ToString();
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestUserEquals()
{
Assert.IsFalse(user.Equals(new UserTeacher(1, "111", "222")));
}
[TestMethod]
public void TestUserHash()
{
Assert.AreEqual(user.GetHashCode(), 1);
}


[TestMethod]
public void TestTestToString()
{
var except = "Экзамен Math exam 16.06.2022";
var actual = test.ToString();
Assert.AreEqual(except, actual);
}
[TestMethod]
public void TestTestEquals()
{
Assert.IsFalse(test.Equals(new Credit(1, "Math exam", new DateTime(2022, 06, 16), 13, 1)));
}
[TestMethod]
public void TestTestHash()
{
Assert.AreEqual(test.GetHashCode(), 13);
}
}
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Иерархическая схема классов

ПРИЛОЖЕНИЕ Г
(обязательное)
Руководство пользователя

1 Общие сведения

Раздел «Руководство пользователя» содержит материал в обязательном порядке необходимый для ознакомления. В разделе описывается уровень допустимых знаний пользователя, документы, описывающие особенности работы и функционирования системы, необходимое программное обеспечение (если требуется).

Целью разработки программного продукта является автоматизация деятельности предприятий по учёту успеваемости студентов. Результатом разработки является приложение, включающее в себя функции доступа и обработки данных регламентируемых предметной областью проекта.

Пользователь системы должен обладать следующей квалификацией:

–пользовательские навыки работы с ЭВМ;

–знание предметной области и знакомство с руководством пользователя.

Список документов, предоставляемых к обязательному ознакомлению перед началом работы:

–настоящее руководство пользователя;

–инструкциями по технике безопасности (для работы с ЭВМ).

2 Требования к запуску приложения

Для запуска приложения требуется:

–операционная система Windows 7, 8, 10;

–установленный .NET Framework;

–заранее настроенное соединения с базой данных системным-программистом.

3 Запуск приложения

Приложение можно запустить через *IDE Visual Studio*. После запуска программы пользователь должен авторизоваться в системе, затем он сможет продолжить работу с приложением. Для завершения программы требуется нажать на значок меню в правом верхнем углу программы и выбрать соответствующий пункт.

4 Непредвиденные ситуации

Могут возникнуть несколько типов ошибок:
–неверно введенные данные при авторизации;
–некорректно введённые данные;
–потеря соединения с базой данных;
–не удалось изменить запись;
–не удалось добавить запись;
–не удалось удалить запись;
–доступ к данному ресурсу недоступен.

При возникновении любых непредвиденных ситуаций рекомендуется перезапустить приложение. Если ошибка связана с подключением к базе данных обратитесь к системному-программисту. Если после повторного использования программы возникают ошибки сообщите разработчику, или программисту, который в данный момент поддерживает программу данной программы для её устранения.

ПРИЛОЖЕНИЕ Д
(обязательное)
Руководство системного программиста

1 Общие сведения о приложении

Приложение предназначено учета успеваемости студентов во время сессии.

Данное приложение выполняет следующие функции:

–управление данными: добавление, удаление, просмотр, редактирование данных;

–выборка данных для предоставления отчётов;

–получение данных из базы данных;

–предоставление отчётов в формате *XML*;

–Для функционирования программно-аппаратного комплекса необходимо наличие следующих компонент:

–СУБД: *MySQL*;

–на стороне клиента: репозиторий проекта *IDE Visual Studio*.

2 Структура программы

Приложение реализовано на языке *C#*. В качестве хранилища базы данных используется *PostgreSQL*, в качестве технологии доступа к данным применяется *LINQ*.

3 Настройка программы

Для корректной работы приложения требуется предварительно настроить подключения к базе данных, если база данных не создана, то сгенерировать её.

Так как приложения работает с СУБД *MySQL*, требуется предварительно её установить на компьютер с приложением, либо компьютер, находящийся в одной сети с ним.

Далее потребуется настройка соединения с СУБД. Для этого требуется изменить строку подключения для уже существующей СУБД.

4 Проверка программы

Для проверки программы на работоспособность требуется заново провести модульное и интегрированное тестирование для проверки:

−на соединения с базой данных;

−на работоспособность бизнес-логики и возникновения исключительных

ситуаций;

−корректность вывода данных и представления данных.

5 Дополнительные возможности

В качестве дополнительных возможностей можно выделить:

–многопользовательский доступ;

–гибкая настройка СУБД;

–возможность редактирование личной информации пользователя;

–валидация введённых данных.

6 Сообщение системному программисту

В приложении могут возникать следующие системные ошибки:

−не удалось установить подключения к базе данных;

−не удалось сохранить *XML*-отчёт;

−не удалось добавить элемент.

Для устранения данных ошибок требуется перезапустить программу, если ошибки не исчезли требуется проверить корректность введённых данных и соединения с базой данных. Если ошибки так и не удалось устранить требуется обратится к разработчику программы, либо программисту в данный момент её поддерживающий.

ПРИЛОЖЕНИЕ Ж
(обязательное)
Руководство программиста

1Назначение и условия применения программы

Программное средство предназначено для учёта успеваемости студентов во время сдачи сессии за время обучения в ВУЗе.

Основные функции приложения:

–предоставление отчётов об успеваемости студентов;

–получение информации из базы данных;

–управление данными о факультетах;

–управление данными о специальностях;

–управление данными о группах;

–управление данными о студентах;

–управление данными о испытаниях;

–управление данными пользователей;

–управление пользователями системы;

–управление ролями пользователей.

Для запуска приложения должен быть установлен *.NET Framework*. Для считывания данных устройство должно соединятся с базой данных *MySQL*.

2 Характеристики программы

Разработанное приложение написано на языке программирования *C#* в среде разработке *Visual Studio* 2022.

Для хранения информации используется база данных *MySQL*. Работа с ней осуществляется с помощью доступа к данным *LINQ*, работающая на основании стандартных драйверов для подключений *ADO*.

3 Обращение к программе

Обращение к программе происходит с помощью среды разработки, например, *Visual Studio* 2022 и настройки соединения с СУБД *MySQL*.

Для дополнения программного обеспечения новым функционалом можно использовать любую среду разработки на языке программирования *C#*. Продукт покрыт автоматизированными функциональными тестами и их использование позволяет изменять исходный код без опасности нарушить какую-либо часть существующего функционала.

4 Входные и выходные данные

В качестве входных данных может использоваться тестовый набор данных, он генерируется с помощью *SQL*-скрипта приложенным в приложении, базу данных и тестовые записи.

Выходными данными для приложения являются сгенерированные *XML*-отчёты.

5 Сообщение в ходе работы приложения

При работе программа может оповещать пользователя о следующих неполадках в работы системы:

– некорректные учетные данные пользователя;

– несуществующий пользователь;

– ошибки корректности ввода данных.

Данные сообщения передаются в специальном виде ошибки с кодом ошибки и описанием.