Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет

Кафедра экономической информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу:

«Объектно-ориентированное программирование»

на тему:

«РАЗРАБОТКА ПРОГРАМНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ОРГАНИЗАЦИИ РАБОТЫ ПО КУПЛЕ-ПРОДАЖЕ ВИДЕОКОНТЕНТА»

Студент гр. 924401 Кондрашов Д. А.

Руководитель старший преподаватель Салапура М.Н.

каф. эконом. информ. БГУИР

Минск 2021**СОДЕРЖАНИЕ**

Введение 5

1 Обзор по купле-продаже видеоконтента 7

* 1. Описание приложений по купле-продаже видеоконтента 7
  2. Программные аналоги 8
  3. Эффект от автоматизации купли-продажи видеоконтента 9
  4. Выводы 10

1. Функциональное моделирование на основе стандарта IDEF0 11
2. Структура используемых данных 20
3. Разработка и описание диаграмм классов приложения 21
4. Разработка и описание диаграммы вариантов использования

приложения 27

1. Схема алгоритма работы всей программы и алгоритма работы двух и более основных методов 28
   1. Схема алгоритма работы всей программы 28
   2. Схемы алгоритмов нескольких основных методов 28
2. Описание алгоритма запуска приложения, его использования, результаты работы программы, тестирования обработки ошибок 29

Заключение 34

Список использованных источников 35

Приложение А Листинг кода 36

Приложение Б Диаграмма классов 56

Приложение В Диаграмма вариантов использования 57

Приложение Г Схемы алгоритмов 58

# ВВЕДЕНИЕ

В данный момент становится очень востребовано онлайн образование и это не только онлайн занятия. В век информационных технологий стало очень популярно обучение с помощью **курсов**, который уже записаны и находятся **на каком-либо ресурсе**.

В 2019 году рынок дополнительного b2c-образования в России [составил](https://research.edmarket.ru/) 38 млрд рублей. В нём есть большие и маленькие сегменты. Самые большие – обучение английскому и программированию. Крупные – фитнес, здоровье, психология, личностный рост, астрология, обучение профессиям. В любом из сегментов можно заработать много. Каждый пятый проект, который смог в них запуститься, зарабатывает около миллиона в месяц, но есть и те, кто пытались, но не смогли, и их тоже немало [1].

Самое популярное направление для обучения – онлайн школы и тренинги. Если с онлайн школами всё достаточно легко – создать сайт на CRM основе, запустить рекламу и всё, то с тренингами сложнее. Обычно тренинги идут с инстаграмма, это может быть: создание личного бренда, правильное оформление фотографий, SMM, администрирование аккаунтов звёзд. Людей, собравшихся становится перечисленными онлайн специалистами – достаточно много. И чтобы данная профессия была достаточно востребованной и не имела слишком много конкуренции, создатели **ограничивают** наборы на тренинги.

Это отличный способ разработчикам получить достаточно быстро отличные деньги из-за наплыва и спроса курсов по данной профессии. Такие тренинги обычно набирают команду из нескольких человек, дают доступ к пред просмотрами данных роликов и после показывают цену за полный доступ к одному или нескольким видеозаписям.

Обычно используют несколько платформ для такого типа подачи видеоконтента:

* GetCourse;
* Антитренинги;
* AlfaCRM.

В функционирование приложений данных платформ входит:

* доступ к порталу;
* предоставление списка видеоконтента;
* оформление заказов.

Разработка приложения для данных процессов, существенно ускоряет продажу и позволяет пользователям не утруждать себя в сложности оформления каких-либо других документов.

Цель курсового проекта – разработка объектной модели и её программная реализация на языке C++.

Задачи курсового проекта:

* обзор приложений по купле-продаже видеоконтента, программных аналогов, методов и алгоритмов решения поставленной задачи;
* функциональное моделирование на основе стандарта IDEF0;
* структура используемых данных;
* описание созданных программных конструкций;
* разработка и описание диаграммы классов приложения;
* разработка и описание диаграммы вариантов использования приложения;
* схема алгоритма работы всей программы и алгоритма работы двух и более основных методов;
* описание алгоритма запуска приложения, его использования, результаты работы программы, тестирования обработки ошибок.

# 1 ОБЗОР СИСТЕМЫ ПО КУПЛЕ-ПРОДАЖЕ ВИДЕОКОНТЕНТА

## **1.1 Описание приложений по купле-продаже видеоконтента**

Обычно, для корректной организации купли-продажи видеоконтента требуется наличие как минимум трех сторон: клиента, гостя и администратора.

Администратор – специалист, отвечающий за добавление контента на платформу, регулирование объёма и редактирование видеоконтента.

Обязанности администратора во многом зависят от той отрасли в которой работает данный специалист. Например, в ресторане администратор должен встретить гостей и создавать для них благоприятную обстановку.

Первоначально понятие "администратор" применялось исключительно в качестве определения должности. Позднее администрирование стало одной из отраслей менеджмента, а специальность "администратор" была выделена в самостоятельную профессию.

На сегодняшний день существует больше десяти узких специализаций администратора, в том числе, администратор систем и аккаунтов в социальных сетях.

Поскольку должность администратора относится к руководящим позициям, спектр обязанностей данного сотрудника довольно обширный. Однако в приложении данного курсового проекта функции администратора ограничены следующими:

* Добавление видео в каталог;
* Удаление видео из каталога;
* Поиск видео в каталоге;
* Редактирование видео в каталоге;
* Сортировка видеоконтента по названию;
* Вывод меню с использованием фильтров.

Гость – тот, кто посещает, что-либо с целью ознакомления. В программе гость имеет несколько функций, он может просматривать каталог и искать нужное ему видео.

В настоящее время клиент считается «главным» пользователем, поскольку именно для него создаются подобные приложения. Сервисы по купле-продаже видеоконтента должен обеспечить пользователю приятное времяпровождение и изучение материала.

В приложении данного курсового проекта клиент имеет простейший функционал:

* Просмотреть каталог;
* Добавить в корзину видеозаписи;
* Пополнить баланс своего аккаунта;
* Искать нужные видеозаписи.

После добавления видеозаписей в корзину, пользователь может просмотреть товары из корзины и купить выборочно или сразу всё.

## **1.2 Программные аналоги**

**Антитренинги.ru –** это универсальная платформа для дистанционного обучения с удобным интерфейсом и множеством функций для создания онлайн курса, открытия онлайн-школы или корпоративного университета. Платформа удобна своим функционалом и низким порогом вхождения в интерфейс, с помощью которого пользователи любого возраста могут без проблем освоиться.

Обычно платформой пользуются преподаватели, коучи, эксперты, которые хотят упаковать свой опыт в онлайн-курс и продавать его. Также этот сервис подойдёт для инфобизнесменов.

Цены на Антитренингах зависят от количества учеников и места на сервере. Минимальный тариф на февраль 2021 года — от 1500 ₽ в месяц. За эти деньги вы можете обучать 2000 учеников и хранить на сервисе 30 Гб учебных материалов. Разделение на тарифы очень важно, так как пользователям может понадобиться разный объём информации. То есть платформа отлично разработана с экономической стороны для одиночный преподавателей и больших компаний [2].

У Антитренингов есть бесплатный пробный период 14 дней. Он нужен для того, чтобы новые пользователи разобрались с интерфейсом в минимальные сроки.

**GetCourse.ru** – Платформа для продажи и проведения обучений. В отличие от Антитренинги.ru, на данной платформе нету разделение на подписки для пользования. Пользователь покупает один раз доступ и сразу же получает ряд возможностей для своих целей в обороте видеоконтента:

* Вебинары прямо на платформе;
* Брендирование лого, домена и дизайна;
* Обучение.

После покупки, пользователь получает доступ к CRM системе для сегментирования и настройки верного расписания вебинаров или доступа к уже отснятым видеозаписям в определённое время.

Один из факторов успеха – это доступность. У GetCourse присутствует своё мобильное приложение, с помощью которого можно просматривать видеозаписи в любом месте и в любое время.

Преимущества сервисов:

* Удобство интерфейса и обучающие видео уроки;
* Возможность использовать приложение в любом месте и в любое время;
* Мобильное приложение;
* Адекватные цены на тарифы.

Недостатки сервисов:

* Большие ограничения на дешёвые тарифы;
* Возрастное ограничение;
* Невозможность просмотра контента в роли гостя;
* Нету возможности просмотреть топ видеоконтента и курсов.

## **1.3 Эффект от автоматизации купли-продажи видеоконтента**

С помощью автоматизации можно организовать удобный способ организации купли-продажи видеоконтента через сайт и мобильное приложение. Система позволяет редактировать и контролировать новые поступление видеоконтента и организованно его продавать.

Добавление бонусов для постоянных клиентов. Клиенты будут приятно удивлены тем, что разработчики заботятся о своих клиентах.

Интеллектуальная система распределения покупок пользователя, которая учитывает ранние покупки и составляет список рекомендуемых к просмотру и покупке видеозаписей (обучающего видеоконтента)

Онлайн поддержка, с помощью которой в любой момент можно задать вопрос, связанный с технической или какой-либо другой проблемой, также связанной с консультированием по поводу нужного направления в выборе видеоконтента.

Анализ самого востребованного видеоконтента и продвижение его в топ.

Исходя из проведенных исследований следует вывод о целесообразности и актуальности данной темы для проведения проектирования и разработки программного обеспечения для организованной работы по купле-продаже видеоконтента, так как в последнее время наблюдается большой спрос на онлайн обучение. Пользователи после карантина начали понимать, что самообразование – это несложно и доступно для каждого.

# 2 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

Для функционального описания модели работы программы была выбрана такая нотация как IDEF0. Данная функциональная модель представлена в виде набора блоков, каждый из которых в свою очередь представлен в виде чёрных прямоугольных областей с входами и выходами, управлением и механизмами, которые уточняются и раскрываются до необходимого уровня сложности и детальности. Данная модель даёт возможность описать все основные виды процессов, представленные в данной программе.

Наиболее популярной нотацией, используемой для моделирования бизнес-процессов верхнего уровня, является нотация IDEF0. Правила и система обозначений IDEF0 позволяют отобразить структуру и функции любой системы. Еще раз: IDEF0 – нотация, в которой очень удобно показывать структуру системы, структуру бизнес-процесса. И удобно показывать основные функции. Именно поэтому IDEF0 – идеальная нотация для моделирования верхнего уровня бизнес-процессов. По статистике 90% всех описаний процессов имеют верхнеуровневое, функциональное описание в IDEF0, еще 10% - это описание верхнего уровня в BPMN.

**Бизнес-процесс** в обозначениях IDEF0 – прямоугольник (блок), его связи с элементами внешней среды или другими процессами – это стрелки. Это базовый минимум, с которого можно начинать знакомиться с нотацией, пробовать фиксировать в ней какие-то первые процессы «в карандаше». Внутри прямоугольника (блока) вписывается название функции/процесса и его номер.

Декомпозиция − разделение моделируемой функции на функции- компоненты.

Прямоугольник представляет собой функцию (деятельность, процесс, операцию, действие, работу), которая имеет фиксированную цель и приводит к некоторому конечному результату. Имя функции должно быть глаголом или глагольным оборотом, т.е. выражать действие (например, «Изготовить деталь», «Расчет допускаемых скоростей», «Формирование ведомости ЦДЛ № 3»).

Взаимодействие функций между собой и внешним миром описывается в виде стрелок. В IDEF0 различают 5 видов стрелок:

– вход (англ. input) – материальный объект или информация, которые используются и преобразуются функцией для получения результата (выхода). Вход отвечает на вопрос «Что подлежит обработке?». В качестве входа может быть, как материальный объект (сырье, деталь, экзаменационный билет), так и нематериальный (запрос к БД, вопрос преподавателя). Допускается, что функция может не иметь ни одной стрелки входа. Стрелки входа всегда рисуются входящими в левую грань функции;

– управление (англ. control) – управляющие, регламентирующие и нормативные данные, которыми руководствуется функция. Управление отвечает на вопрос «Что вызывает или регламентирует выполнение функции?». Управление влияет на функцию, но не преобразуется ей, т.е. выступает в качестве предписания или ограничения. В качестве управления могут быть стандарты, нормативы, правила, расценки, устные указания. Стрелки управления рисуются входящими в верхнюю грань функции. Если при построении диаграммы возникает вопрос, как правильно нарисовать стрелку сверху или слева, то рекомендуется ее рисовать как вход (стрелка слева);

– выход (англ. output) – материальный объект или информация, которые представляют результат выполнения функции. Выход отвечает на вопрос «Что является результатом выполнения функции?». В качестве выхода может быть как материальный объект (деталь, автомобиль, платежные документы, ведомость), так и нематериальный (выборка данных из БД, ответ на вопрос, устное указание). Стрелки выхода рисуются исходящими из правой грани функции;

– механизм (англ. mechanism) – ресурсы (средства), которые задействованы при выполнении функции. Механизм отвечает на вопрос «Кто выполняет функцию или посредством чего?». В качестве механизма могут быть персонал предприятия, студент, станок, оборудование, программа. Стрелки механизма рисуются входящими в нижнюю грань функции [3].

На рисунке 2.1 представлена контекстная диаграмма верхнего уровня.

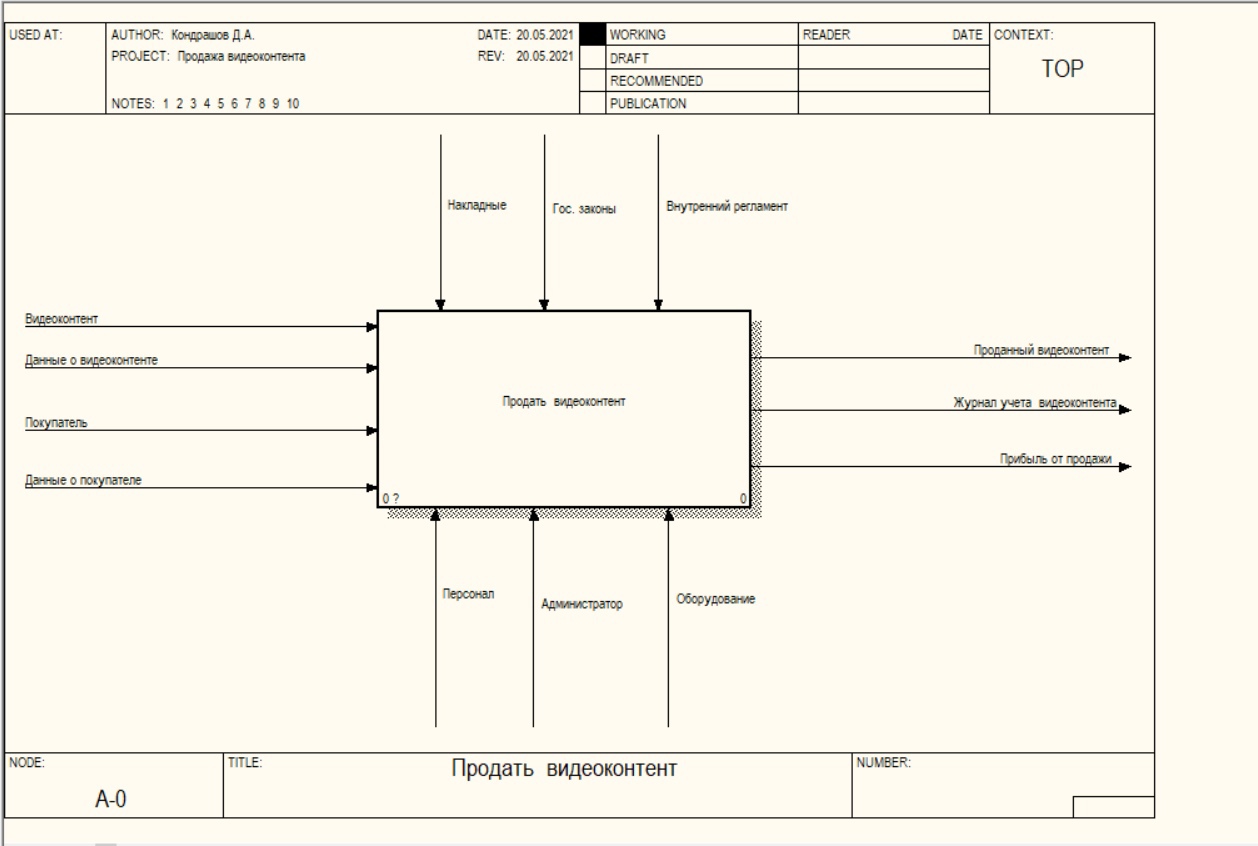


Рисунок 2.1 - Контекстная диаграмма верхнего уровня

Основной блок – это «Продать видеоконтент».

Входные данные – «Видеоконтент», «Данные о видеоконтенте», «Покупатель», «Данные о покупателе».

В качестве управления выступают «Накладные», «Государственные законы», «Внутренний регламент».

Механизмом служат «Персонал», «Администратор» и «Оборудование».

На выходе имеем «Проданный видеоконтент», «Журнал учёта видеоконтента» и «Прибыль от продажи».

На рисунке 2.2 изображена декомпозиция контекстной модели.

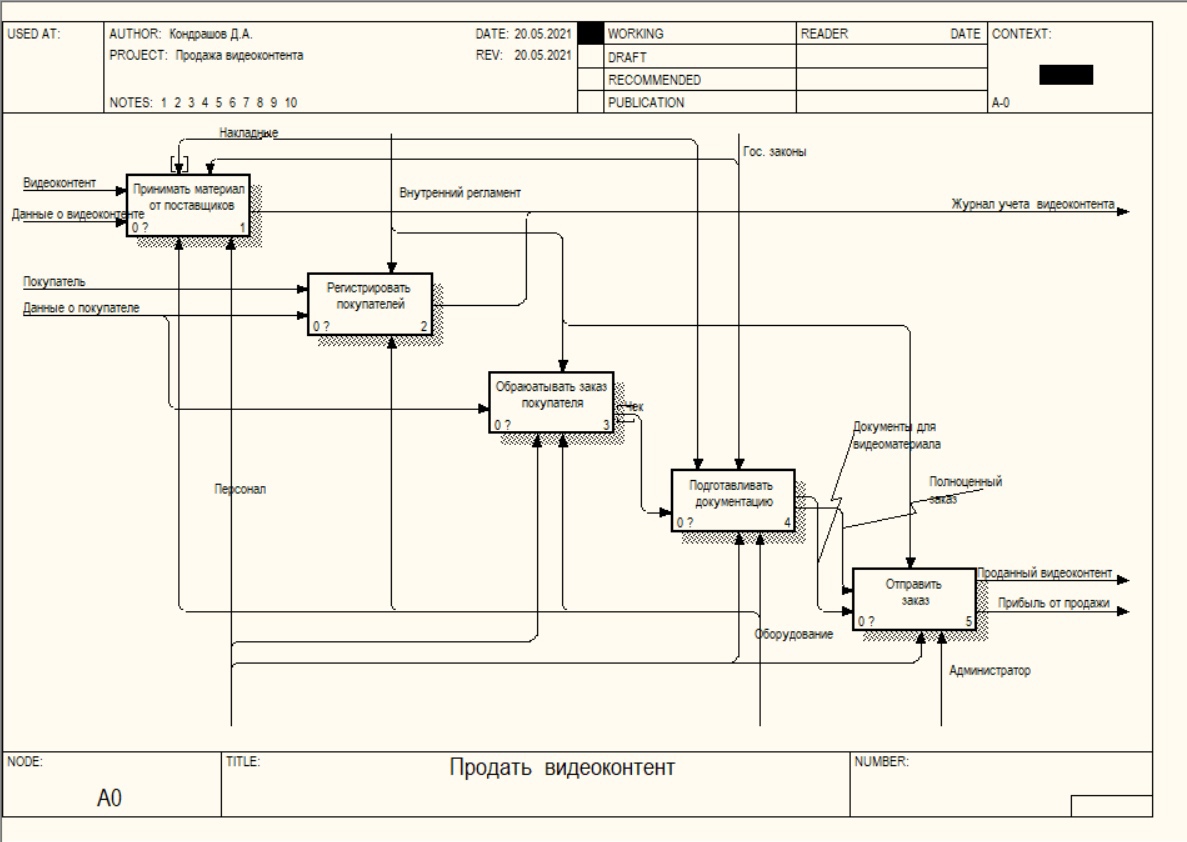


Рисунок 2.2 – Декомпозиция контекстной модели

Первый компонент «Принимать материал от поставщиков» значит, что перед тем, как пользователи смогут воспользоваться программой, нужно заполнить контентом всю систему.

Второй компонент «Регистрация покупателей» пользователей перед покупкой обязательно нужно зарегистрироваться. На это влияют разные факторы: подборка видеоконтента по предпочтениям для каждого пользователя, привязка платёжной карточки.

Третий компонент «Обработать заказ покупателя» происходит после выполнения предыдущих двух и означает, что клиент может выбрать необходимые ему видеозаписи и оформить свой заказ.

Четвёртый компонент «Подготовить документацию» говорит о том, что, после формирования заказа, клиенту представится возможность просмотреть свой заказ, оплатить его и согласиться с предоставленной документацией о возможностях возврата.

Пятый компонент «Отправить заказ» говорит о том, что заказ уже сформирован и готов для передачи клиенту.

На рисунке 2.3 представлена декомпозиция компонента «Принимать материал от поставщиков».

Компонент включает в себя 4 блока: «Проверка видеоконтента», «Замена брака», «Подписать документы по примеру», «Внести данные в базу».

В качестве входной информации используем «Видеоконтент» и «Данные о видеоконтенте». Проверка видеоконтента требуется для исключения неправильно товара от поставщиков и последующей его замене. После блока «Заменить брак» имеем на выходе «Отобранные видеоконтент» и переходим к оформлению соответствующих документов о купле-продаже товара для исключения проблем возврата. «Документация о видеоконтенте» сразу поступает в базу данных. В результате получаем Журнал учёта видеоконтента.

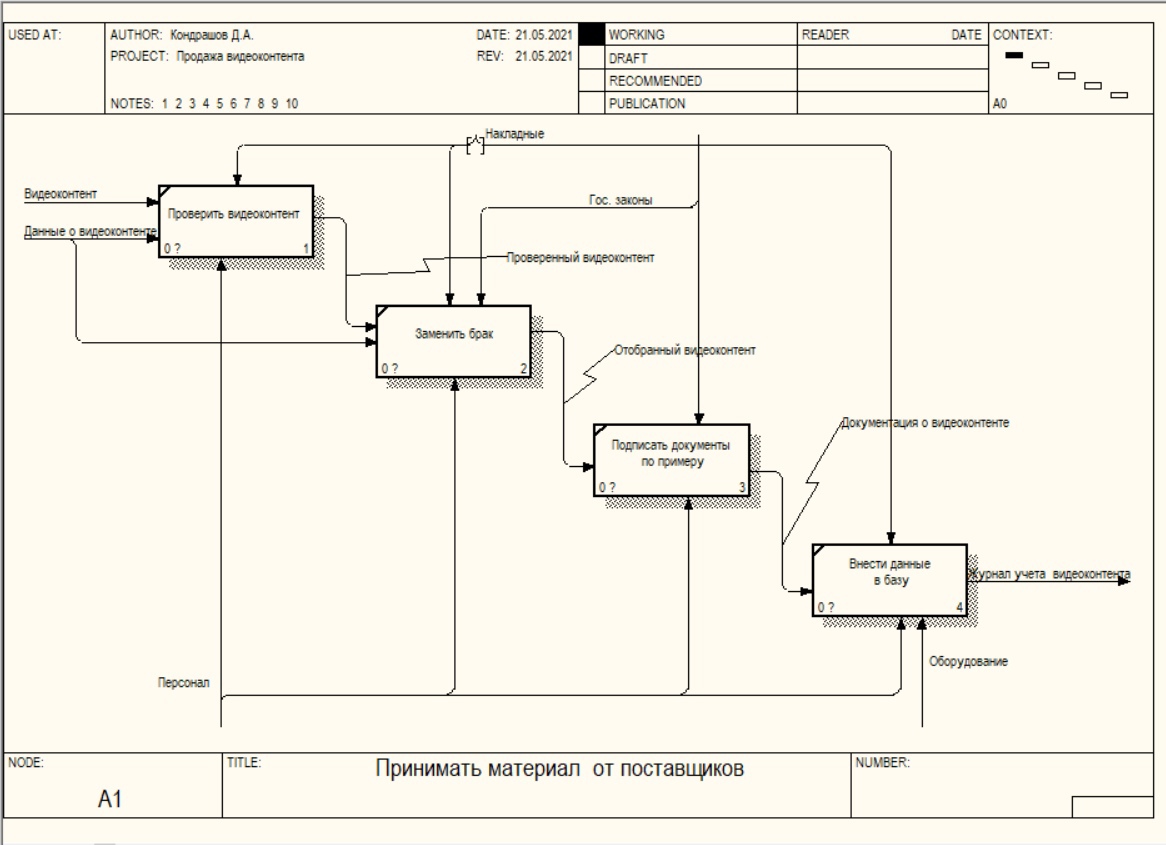
****

Рисунок 2.3 – Декомпозиция компонента «Принимать материал от поставщиков»

На рисунке 2.4 представлена декомпозиция компонента «Регистрировать покупателей».

Компонент включает в себя 3 блока: «Получить данные о покупателе», «Закрепить за ним учётную запись», «Добавить в свою документацию».

В качестве входной информации используем Покупателя и данные о покупателе. С помощью механизма «Оборудование» или программного обеспечения получаем данные для аккаунта из базы данных. С помощью данных для аккаунта из базы данных, можно закрепить учётную запись за пользователем. Аккаунт пользователя вносится в документацию и на выходе получаем «Журнал учёта видеоконтента». В дальнейшем зарегистрированный пользователь имеет возможность перейти к корзине и произвести заказ.

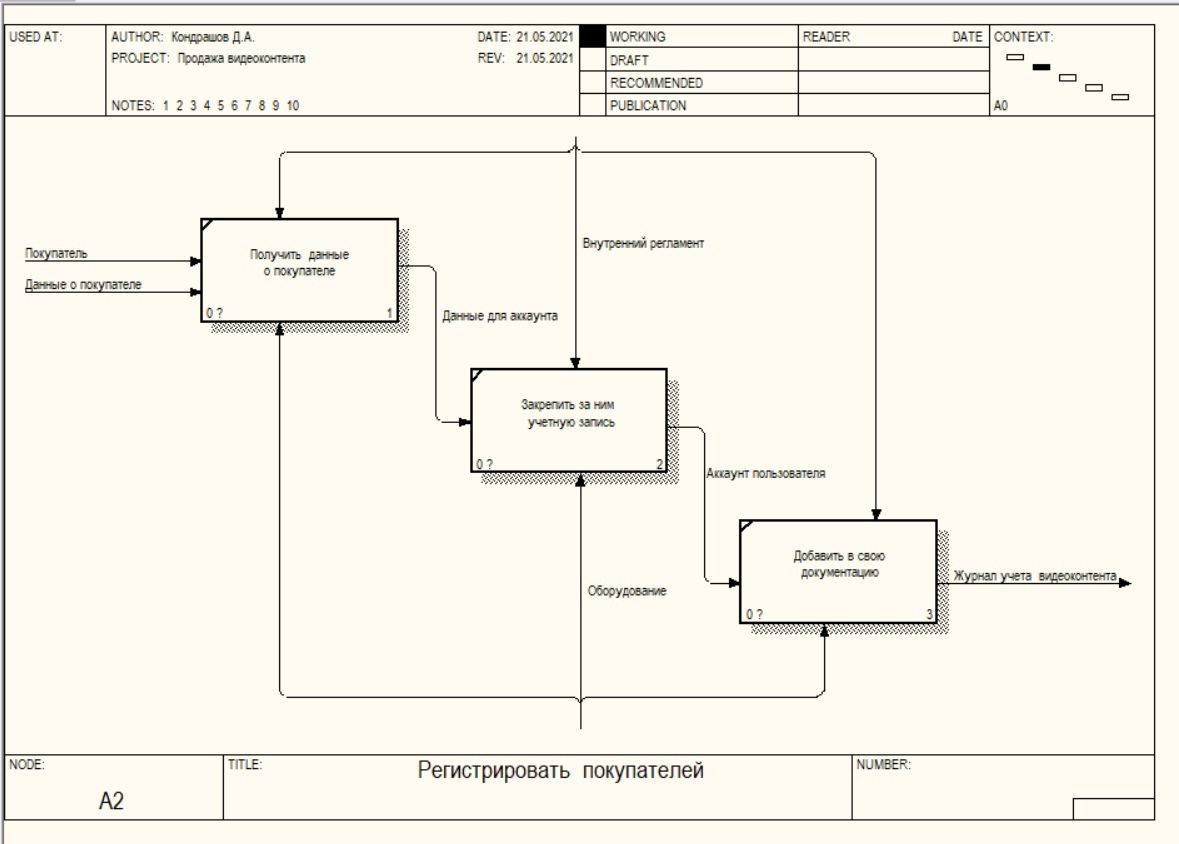
****

Рисунок 2.4 – Декомпозиция компонента «Регистрировать покупателей»

На рисунке 2.5 показана декомпозиция задачи «Обрабатывать заказ покупателей».

Компонент включает в себя 3 блока: «Получить данные о заказе», «Произвести оплату» и «Добавить в журнал учёта».

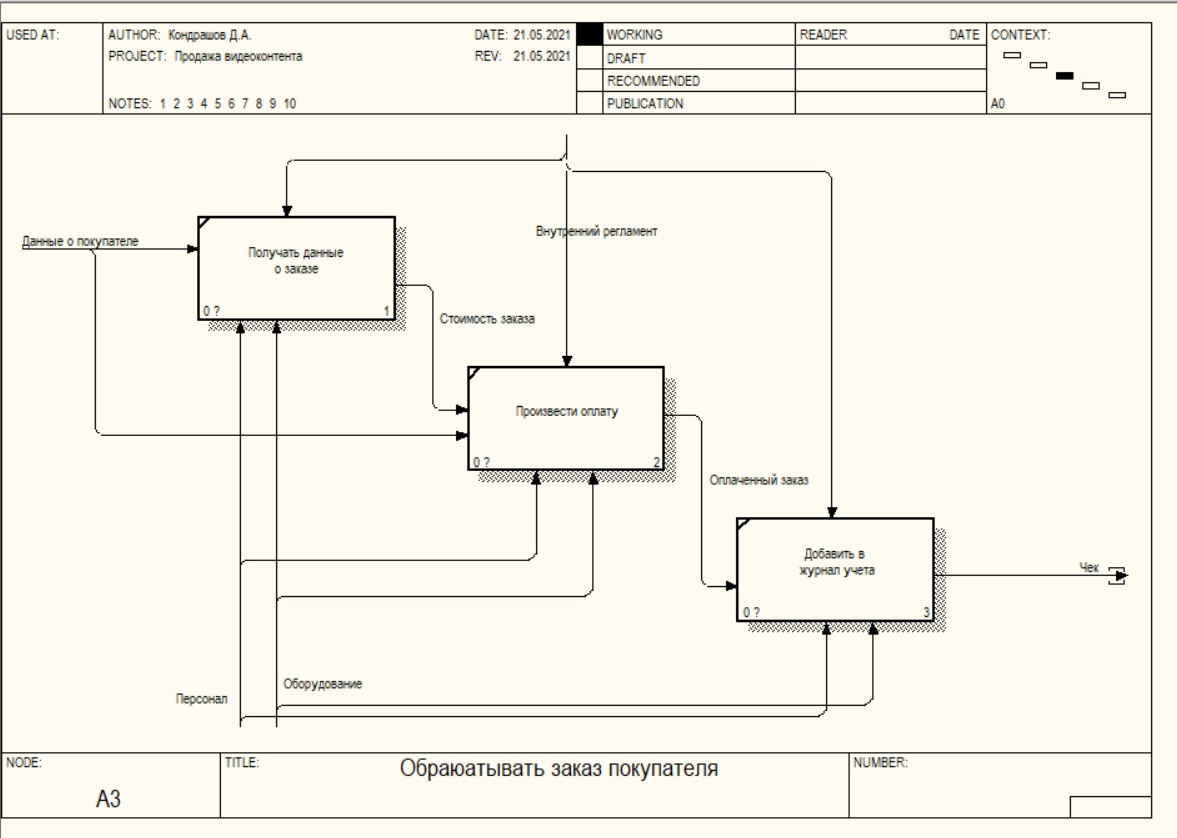


Рисунок 2.5 – Декомпозиция компонента «Обрабатывать заказ покупателя»

Во входных данных текущего компонента используется «Данные о покупателе». После того, как клиент зарегистрировался и сформировал свой заказ оказывается в «Обработке заказов». Клиент получает данные о сформированном заказе (сверяет стоимость и контент) и переходит к следующему блоку «Произвести оплату». Все действия производятся на платформе с помощью Оборудования. Когда на аккаунте присутствует необходимая сумма можно приступать к оплате. Оплаченный заказ добавляется в журнал учёта по купле-продаже видеоконтента. После выполнения всех блоков на выходе клиент получает чек на электронную почту.

На рисунке 2.6 представлена декомпозиция компонента «Подготавливать документацию».

Компонент включает в себя 2 блока: «Свериться с журналом учёта», «Выписать гарантию».

В качестве входной информации поступает «Чек». Управляющий механизм - «Накладные» и «Государственные законы». Механизмы – «Персонал» и «Оборудование». В результате, на выходе получаем «Видеоконтент», «Полноценный заказ».

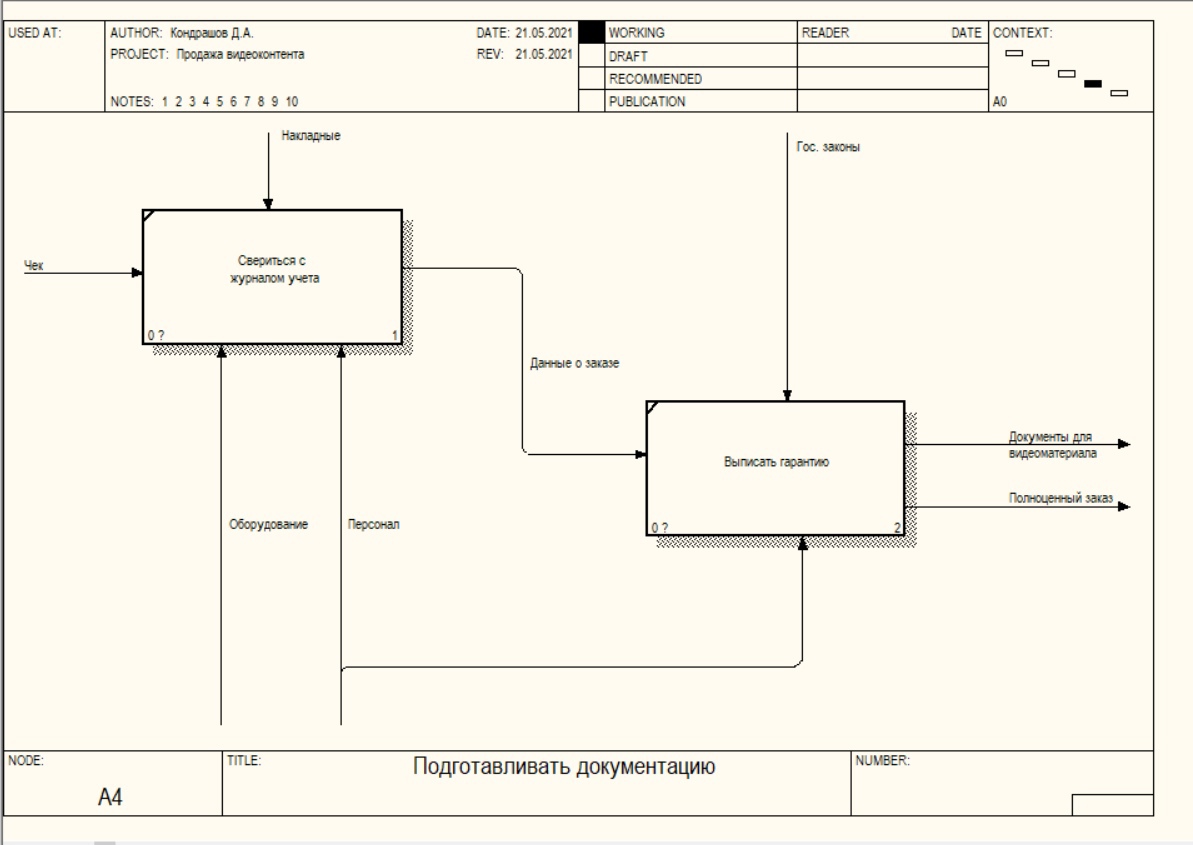


Рисунок 2.6 – Декомпозиция компонента «Подготавливать документацию»

На рисунке 2.7 представлена декомпозиция компонента «Подготавливать документацию».

Компонент включает в себя 3 блока: «Найти свободного сотрудника», «Передать данные сотруднику» и «Отправить видеоматериал заказчику».

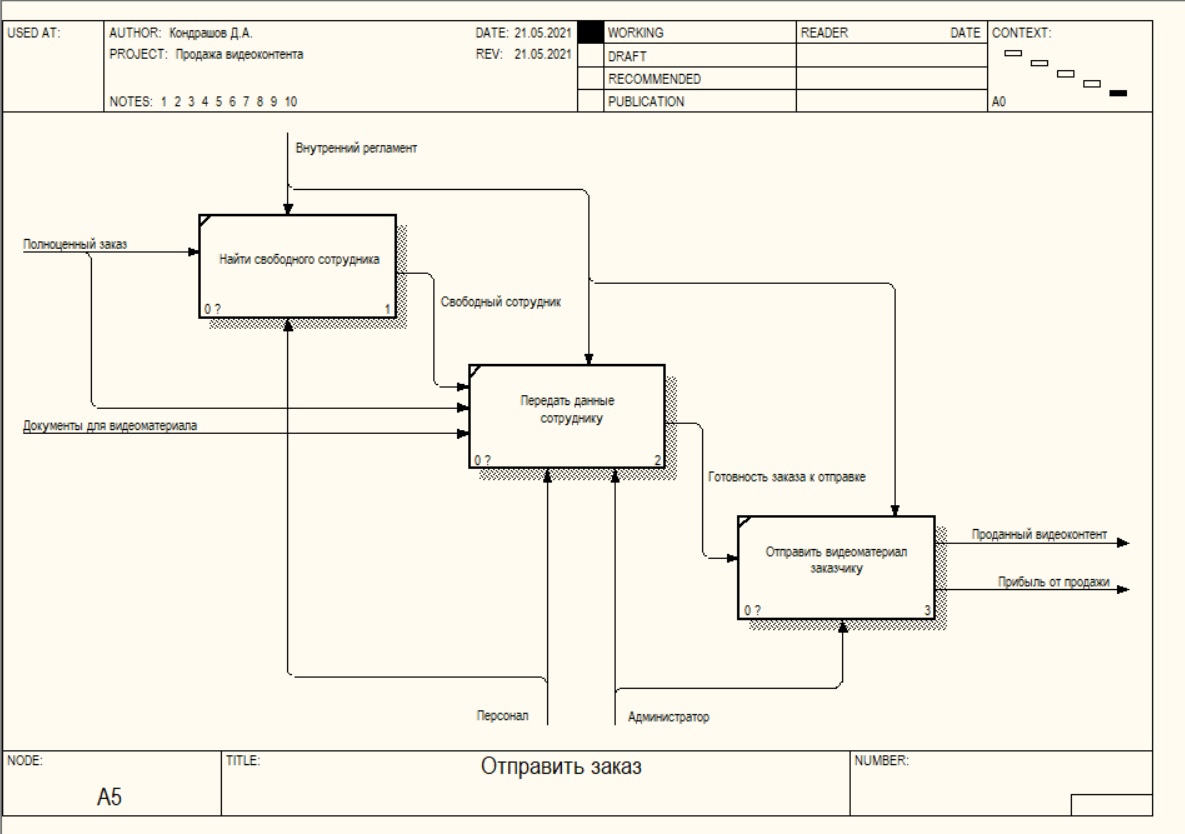


Рисунок 2.7 – Декомпозиция компонента «Отправить заказ»

Полученный заказ поступает на вход к сотрудникам. После нахождения свободного сотрудника, данные отправляются к свободному сотруднику. Свободный сотрудник сообщает о «Готовности заказа к отправке» и отправляет видеоматериал заказчику. На выходе получаем «Проданный видеоконтент» и «Прибыль от продажи».

**3 СТРУКТУРА ИСПОЛЬЗУЕМЫХ ДАННЫХ**

Все данные программы хранятся в соответствующих файлах в папке проекта. Всего для работы программы используется 3 файла: «Admins.txt», «Customers.txt», «Videoo.txt». Для чтения и записи данных в файлы используются функции библиотеки stdio.h fread() и fwrite() соответственно.

Файлы «Admins.txt», «Customers.txt» хранят данные об администраторах и клиентах соответственно. Под данными подразумевается:

* Имя;
* Логин для входа;
* Пароль для входа (в зашифрованном виде);
* Остаток денежных средств на балансе аккаунта (у администраторов данный пункт отсутствует).
* Купленный видеоконтент (у администраторов данный пункт отсутствует).

Всякий как раз после регистрации пользователя любой из вышеуказанной роли в конец соответствующего файла добавляется информация о новом пользователе.

Файл «Videoo.txt» хранит список всего видеоматериала вне зависимости от доступности.

Формат данных:

* Наименование;
* Автор;
* Жанр;
* Стоимость;
* Количество возможных покупок видеозаписи.

**4 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММ КЛАССОВ**

**ПРИЛОЖЕНИЯ**

Объектно-ориентированное программирование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования. Именно по такой технологии был создан данный проект.

В проекте были разработаны шесть классов и одна структура: класс Account, от которого наследуются публично классы Customer, Admin, Guest; Класс Store для просмотра каталога и управления потоком видеоконтента; Класс Video для добавления нового видеоконтента и управлением старым; Структура VideoStatus для перегрузки операторов ввода-вывода и отслеживания уже добавленных видеозаписей. Диаграмма классов проекта представлена в приложении Б.

В данном проекте были реализованы основные принципы объектно-ориентированного программирования: инкапсуляция, наследование и полиморфизм.

Наследование является способом расширения функциональных возможностей существующих типов данных. Например, типов, определённых как часть библиотеки. Есть правило, что менять уже существующее не годится. Переписывать всё заново тоже плохо, потому что частично нужная функциональность уже реализована. Её нужно только дополнить и расширить [4].

Инкапсуляция (от лат. in capsule – в оболочке) – это заключение данных и функционала в оболочку. В объектно-ориентированном программировании в роли оболочки выступают классы: они не только собирают в одном месте переменные и методы, но и защищают их от вмешательства извне (сокрытие). Очень важно то, что инкапсуляция не является сокрытием. Сокрытие – это часть инкапсуляции. Основным использованием инкапсуляции в проекте являются модификаторы доступа: private, protected и public [5].

Полиморфизм характеризуется следующей фразой: «один интерфейс – множество методов». Попросту говоря, полиморфизм представляет собой атрибут, который позволяет использовать один и тот же интерфейс при реализации целого класса различных действий. Выбор того, какое именно действие будет совершено, определяется конкретной ситуацией.

В данном проекте были реализованы три описанных выше принципа объектно-ориентированного программирования.

Класс Account содержит общие поля для всех ролей программы: \_type, \_store, \_password, name. Класс содержит две дружественные функции для перегрузки оператора вывода и сохранения данных в файлы. Также класс содержит 7 методов, которые можно увидеть на рисунке 4.1.

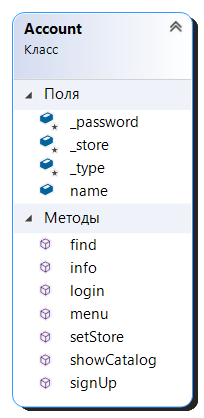


Рисунок 4.1 – Вид класса Account

Класс Guets является наследником класса Account и имеет только два метода для вызова меню гостя и получения информации о текущем статусе аккаунта. Вид класса изображен на рисунке 4.2.

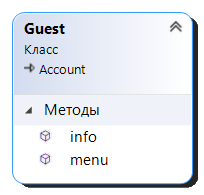


Рисунок 4.2 – Вид класса Guest

Класс Admin является наследником класса Account и имеет только 7 методов для управления видеоконтентом и 3 дружественные функции для работы с записью в файл, две из которых перегрузка операторов ввода-вывода. Класс Admin также имеет своё меню для управления и переопределённый метод info для получения статуса текущего аккаунта. Класс используется для добавления видеоконтента в каталог. Вид класса изображен на рисунке 4.3.

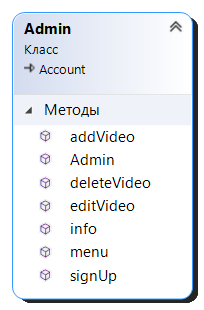


Рисунок 4.3 – Вид класса Sort

Класс Сustomer используется для инициализации клиента и содержит 4 защищенных поля, которые описывают каждого клиента: \_crate указатель корзину покупок, money – деньги для добавления к аккаунту, totalPrice – конечное количество денег для покупок. В классе есть 8 методов основные из которых, для работы с корзиной: addToCrate – для добавления элемента в корзину покупок, buy для покупки из корзины, Customer – для инициализации пользователя, showCrate – просмотр корзины покупок, editVideo – для изменения свойств видеозаписей в каталоге и метод singUp – для входа в аккаунт. Диаграмму класса можете наблюдать на рисунке 4.4.

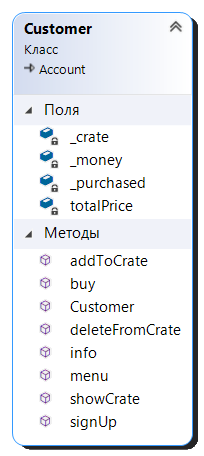


Рисунок 4.4 – Вид класса Customer

Класс Video хранит 4 приватных поля для определения свойств видеоконтента и 13 методов для управления видеоконтентом и 3 из которых – дружественные функции для перегрузки операторов. Методы author, genre (+1 перегрузка), hasGenre, name, price, setAuthor, setGenre, setName, setPrice нужны для создания каких-либо свойств у нового видео и внесения его в каталог всего видеоконтента. Метод Video нужен для создания начальных свойств видеоконтента. Перегруженные операторы сравнения, нужны для проверки на повторения основных свойств и при возвращении false, вызывается исключение, и просьба повторить ввод. Вид класса изображен на рисунке 4.5.

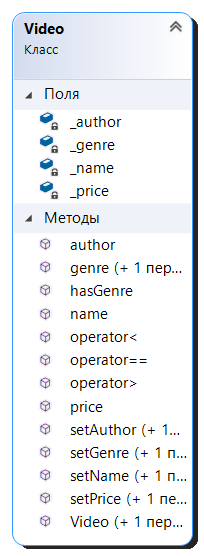


Рисунок 4.5 – Вид класса Video

Класс Store содержит приватное поле для количества видеозаписей доступных для покупки на платформе и указатель на список всего видеоконтента. Класс Store отвечает за управление, поиск, удаление, редактирование всего видеоконтента. Класс содержит 12 методов. Основные из них: sort – сортировка доступных видеозаписей в каталоге по названию, findVideo – поиск доступных видеозаписей по названию или автору. Вид класса можно увидеть на рисунке 4.6.

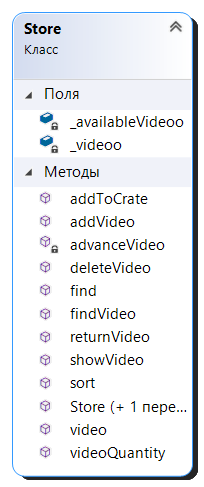


Рисунок 4.6 – Вид класса Store

Структура VideoStatus содержит публичные поля isAvailable – булевая переменная для указания доступа и запрета к покупке определённого видео, quantity – ограничение на количество покупки определённого контента, video – указатель на видеозапись. И метод VideoStatus – для проверки статуса определённой видеозаписи. Также присутствуют 3 перегрузки операторов ввода-вывода для манипуляций с файлами.

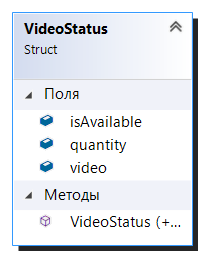


Рисунок 4.7 – Вид структуры VideoStatus

**5 РАЗРАБОТКА И ОПИСАНИЕ ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ**

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (действующим лицом, актантом, актёром) называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования – это спецификация сервисов (функций), которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

В процессе разработки данного приложения была создана диаграмма вариантов использования.

В приложении B на рисунке 1, представлена диаграмма вариантов использования данной системы.

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Они предназначены для упрощения взаимодействия с конечными пользователями системы, и, так же, пригодятся для определения необходимых характеристик системы. Обобщая, диаграмма вариантов использования показывает, что должна делать система, не указывая на применяемые методы.

Варианты использования (usecase) описывает, с точки зрения действующего лица, группу действий в системе, приводящих к конкретному результату.

Действующее лицо (actor) является внешним источником, взаимодействующим с системой и может быть, как реальным человеком, так и компьютерной системой. Действующие лица представляют не физических людей, а их роли.

**6 СХЕМА АЛГОРИТМА РАБОТЫ ВСЕЙ ПРОГРАММЫ И АЛГОРИТМА РАБОТЫ ДВУХ И БОЛЕЕ ОСНОВНЫХ МЕТОДОВ**

**6.1 Схема алгоритма работы всей программы**

При запуске программы пользователь может создать новый аккаунт, войти уже в существующий аккаунт или посетить каталог с ограничениями в качестве гостя (без возможности покупки), также может выйти из программы.

В проекте 3 роли: клиент, гость и администратор. Для каждой роли прописан свой функционал. При запуске программы появляется выбор, от которого будет зависеть роль пользователя: 1 – Зайти как покупатель\администратор, 2 – зарегистрироваться как покупатель\администратор, 3 – продолжить как гость. Для выхода из программы, нужно нажать клавишу esc.

Все меню в программе представляют собой нумерованные списки, где выбор номера функции перенаправит пользователя в интерфейс для работы с выбранной функцией.

Меню для гостя включает в себя 3 основных пункта: 1 – просмотр каталога, 2 – вывести информацию об аккаунте, 3 – выход из аккаунта.

Меню для клиента включает в себя 6 основных пункта: 1 – просмотр каталога, 2 – работа с корзиной (5 действий), 3 – купить видео из корзины, 4 – изменить имеющееся количество денег, 5 – вывести информацию об аккаунте, 6 – вывести информацию об аккаунте.

Администратор имеет 6 функций: 1 – просмотреть каталог, 2 – добавить видео в каталог, 3 – редактировать видео из каталога, 4 – удалить видео из каталога, 5 – вывести информацию от текущем аккаунте, 6 – выход из аккаунта.

Общая схема алгоритма работы всей программы будет представлена в приложении Г на рисунках Г.1, Г.2, Г.3, Г.4.

**6.2 Схемы алгоритмов нескольких основных методов**

Одним из наиболее часто используемых методов в проекте является метод вывода в консоль видеоконтента из каталога. Сначала в контейнер STL типа List с файла «Videoo.txt» считываются все объекты типа Video, далее проверяется размер, если размер равен нулю, то выводится сообщение о пустом меню и происходит выход из функции. Блок-схема метода вывода каталога представлена в приложении Г на рисунке Г.5.

**7 ОПИСАНИЕ АЛГОРИТМА ЗАПУСКА ПРИЛОЖЕНИЯ, ЕГО ИСПОЛЬЗОВАНИЯ, РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ, ТЕСТИРОВАНИЯ ОБРАБОТКИ ОШИБОК**

Первым делом после запуска программы пользователь сталкивается с главным меню, представленным на рисунке 7.1. С помощью меню пользователь может войти в уже созданный аккаунт, зарегистрироваться, войти как гость и выйти из приложения.

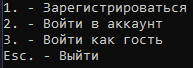


Рисунок 7.1 – Главное меню программы

В случае, если пользователь решил выйти из приложения, консоль очищается и выводит сообщение прощания. Выход из программы изображен на рисунке 7.2.



Рисунок 7.2 – Выход из программы

После нажатия единицу в главном меню, перед пользователем появляется два варианта регистрации: как покупатель и как администратор. В обоих случаях нужно ввести свое имя, затем пароль. Процесс регистрации изображен на рисунке 7.3.

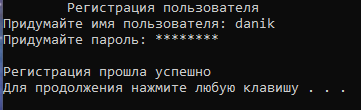


Рисунок 7.3 – Процесс регистрации

После прохождения регистрации пользователь сразу переносится в меню, которое зависит от того, кто был зарегистрирован, покупатель или администратор. Меню выбора роли для покупателя изображено на рисунке 7.4.

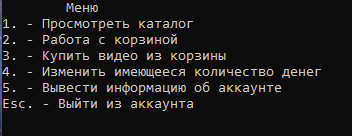


Рисунок 7.4 – Меню покупателя после регистрации

Если был зарегистрирован администратор, то меню будет выводиться уже для администратора с другими функциями. Меню администратора можно увидеть на рисунке 7.5.

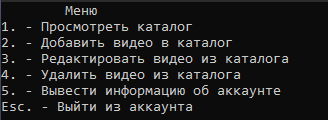


Рисунок 7.5 – Меню администратора

Администратор – единственный, кто может добавлять видеозаписи в каталог. При нажатии на двойку в меню администратора, высвечивается шапка для определённого действия (Добавление видео) и по порядку даются поля для заполнения, которые представлены на рисунке 7.6.

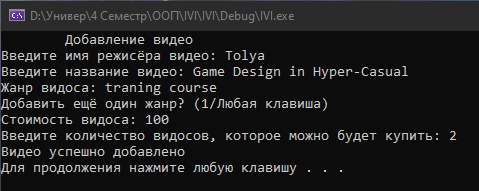


Рисунок 7.6 – Добавление видеозаписей

Просмотр каталога можно осуществить из любого меню, от имени любого пользователя. У всех пользователей это кнопка один. При переходе в Ассортимент магазина, высвечивается соответствующая шапка и весь видеоконтент, вне зависимости от доступности. Также в каталоге присутствуют дополнительные действия для работы со списком: 1 – Поиск видео по запросу; 2 – Сортировка видеоконтента; Esc – Возвращение в предыдущее меню. Ассортимент магазина изображён на рисунке 7.7.

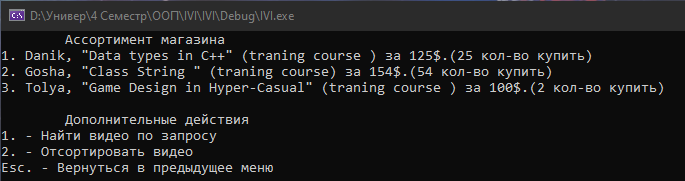


Рисунок 7.7 – Ассортимент магазина

Если администратор захочет удалить какую-либо видеозапись, то сначала ему отобразиться весь каталог, а после предложиться выбрать номер видеозаписи для удаление. Удаление видеозаписи по номеру, показано на рисунке 7.8.



Рисунок 7.8 – Удаление видеозаписи

Также, войти в приложение можно от имени гостя, но с ограниченным функционалом. Гостю предоставляется возможность просмотреть каталог, найти нужную видеозапись и отсортировать весь каталог. Меню гостя изображено на рисунке 7.9.

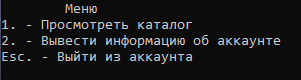


Рисунок 7.9 – Удаление элемента меню

Для каждого пользователя предоставляется возможность просмотра статуса аккаунта. Если в меню информации об аккаунте у гостя и администратора будет написан только текущий статус аккаунта и его функции, то в меню клиента, которое можно увидеть на рисунке 7.10, отображается ещё и количество оставшихся денег, видеозаписи находящиеся в корзине и купленные видеозаписи.

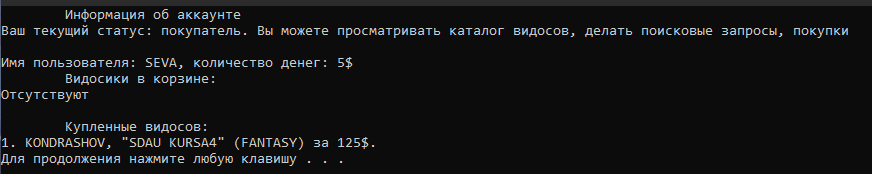


Рисунок 7.10 – Поиск элемента меню

В меню у покупателя, в отличае от гостя и администратора, есть пункт для работы с корзиной – 2. В меню корзины, клиент должен выбрать нужные ему видеозаписи, перед оплатой. Для работы с корзиной есть несколько действий: Добавление видеозаписи из каталога, просмотр выбранных видеозаписей в корзине, удаление видеозаписей из корзины, очистка корзины и переход в предыдущее меню, для покупки всех видеозаписей, находящихся в корзине. Меню для работы с корзиной показано на рисунке 7.11.

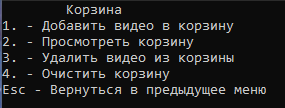


Рисунок 7.11 – Меню для работы с корзиной

Если клиент добавил в корзину несколько видеозаписей и хочет их купить, он должен перейти в начальное меню (рисунок 7.4) и выбрать пункт 3, для покупки видеозаписей из корзины. Пользователь переносится на меню покупки и если у него достаточно денег на балансе, то ему предлагают оплатить все или несколько видеозаписей. Меню покупки видеозаписи из корзины показывается на рисунке 7.12.

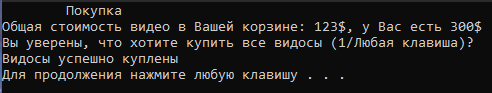
****

Рисунок 7.11 – Меню для работы с корзиной

Если у пользователя недостаточно средств для покупки видеозаписей из корзины, ему высвечивается оповещение о недостаточном количестве средств и колличество средств, которое ему ещё нужно добавить.

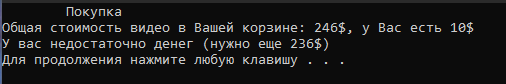


Рисунок 7.12 – Исключение при недостаточном балансе

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была исследована купли-продажи видеоконтента. В результате была создана программа, которую можно использовать уже существующим платформам, связанных с видеохостингом, для расширения профиля услуг.

Интерфейс полученного программного продукта понятен, не требует дополнительных пояснений и защищен от ввода некорректных значений. Для удобства пользователя предусмотрен вывод контекстных сообщений и очистка экрана консоли при переключении на новое действие. Использование файлов для вывода данных позволяет сохранить результаты работы программы даже после завершения ее работы. Используемое в данном проекте шифрование обеспечивает хранение и ввод пароля в зашифрованном виде. Полученный программный продукт соответствует требованиям, поставленным в задании к проекту. Конечный продукт обладает достаточно высокой актуальностью.

В ходе написания программы эффективно использовались основные средства языка С++, позволяющие наиболее эффективно реализовать данный проект и подойти к его разработке с точки зрения методологии объектно-ориентированного программирования.

**Список использованных источников**

[1] MyRouble [Электронный ресурс]. – Режим доступа https://myrouble.ru/chto-takoe-b2b-b2c-b2g-c2c/

[2] Антитренинги [Электронный ресурс]. – Режим доступа https://antitreningi.ru/info/progress/#/login

[3] Методология IDEF0 - Учебная и научная деятельность Анисимова Владимира Викторовича [Электронный ресурс]. – Режим доступа https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6\_2

[4] Блог программиста [Электронный ресурс]. – Режим доступа https://pro-prof.com/forums/topic/oop-inheritance

[5] Инкапсуляция | Программирование на С и С++ [Электронный ресурс]. – Режим доступа http://www.c-cpp.ru/books/inkapsulyaciya

**ПРИЛОЖЕНИЕ А**

**Листинг кода**

Account.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <map>

#include "Account.h"

#include "Store.h"

#include "Crypt.h"

using namespace std;

namespace VideoStore

{

template <class Type>

void showList(list<Type> toShow);

bool accountExists(string name);

void addAccount(Account\* account);

Account\* Account::signUp()

{

cout << "Придумайте имя пользователя: ";

cin >> name;

if (accountExists(name))

{

cout << "Пользователь с таким именем уже существует" << endl;

return nullptr;

}

\_password = enterPassword("Придумайте пароль: ");

addAccount(this);

return this;

}

Account\* Account::login(string name, string password)

{

auto account = \_users.find(name);

if (account != \_users.end())

{

if (account->second->\_password == password)

{

return account->second;

}

}

return nullptr;

}

**Продолжение приложения А**

Account.h

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <map>

#include "Store.h"

#include "Videoo.h"

#include "Crypt.h"

using namespace std;

namespace VideoStore

{

class Account

{

protected:

string \_type;

string \_password;

static Store\* \_store;

public:

string name;

Account\* signUp();

static Account\* login(string name, string password);

virtual void info() = 0;

virtual void menu() = 0;

void showCatalog();

void find();

static void setStore(Store\* store);

friend ofstream& operator<< (ofstream& out, Account& account);

friend void save(Store& store);

};

class Admin : public Account

{

public:

Admin();

Account\* signUp();

void menu();

void addVideo();

**Продолжение приложения А**

void deleteVideo();

void editVideo(VideoStatus\* video);

void info();

friend ifstream& operator>> (ifstream& fin, Admin& admin);

friend ofstream& operator<< (ofstream& out, Admin& admin);

friend void save(Store& store);

};

class Customer : public Account

{

double \_money;

double totalPrice;

vector<Video\*> \_purchased;

vector<Video\*> \_crate;

public:

Customer();

Account\* signUp();

void menu();

void addToCrate(Video\* video);

void deleteFromCrate(int index);

void showCrate();

void buy();

void info();

friend ifstream& operator>> (ifstream& fin, Customer& customer);

friend ofstream& operator<< (ofstream& out, Customer& customer);

friend void load(Store& store);

friend void save(Store& store);

};

class Guest : public Account

{

public:

void menu();

void info();

};

extern map<string, Account\*> \_users;

}

Admin.cpp

**Продолжение приложения А**

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <map>

#include <vector>

#include "Account.h"

using namespace std;

namespace VideoStore

{

Admin::Admin()

{

\_type = "admin";

name = "DefaultAdmin";

\_password = cryptor("1");

}

Account\* Admin::signUp()

{

Admin\* admin = dynamic\_cast<Admin\*>(Account::signUp());

if (admin)

{

ofstream fout("Admins.txt", ios\_base::app);

fout << \*admin;

}

return admin;

}

void Admin::menu()

{

system("cls");

cout << "\tМеню\n1. - Просмотреть каталог\n2. - Добавить видео в каталог\n3. - Редактировать видео из каталога"

<< "\n4. - Удалить видео из каталога\n5. - Вывести информацию об аккаунте\nEsc. - Выйти из аккаунта" << endl;

while (true)

{

char ch = \_getch();

switch (ch)

{

case '1':

{

showCatalog();

break;

**Продолжение приложения А**

}

case '2':

{

addVideo();

break;

}

case '3':

{

system("cls");

\_store->showVideo();

editVideo(\_store->video(enterIndex("Введите номер видео для редактирования: ", \_store->videoQuantity())));

break;

}

case '4':

{

system("cls");

showCatalog();

deleteVideo();

break;

}

case '5':

{

system("cls");

info();

break;

}

case 27:

return;

default:

continue;

}

system("pause");

system("cls");

cout << "\tМеню\n1. - Просмотреть каталог\n2. - Добавить видео в каталог\n3. - Редактировать видео из каталога"

<< "\n4. - Удалить видео из каталога\n5. - Вывести информацию об аккаунте\nEsc. - Выйти из аккаунта" << endl;

}

}

Crypt.cpp

#include <iostream>

#include <conio.h>

**Продолжение приложения А**

#include <fstream>

#include <string>

#include <map>

#include <vector>

#include "Crypt.h"

#include "Store.h"

using namespace std;

namespace VideoStore

{

string cryptor(string toCrypt)

{

string cryptor = "Danik";

int cryptorSize = cryptor.size();

int toCryptSize = toCrypt.size();

for (int i = 0; i < toCryptSize; i++)

{

toCrypt[i] ^= cryptor[i % cryptor.size()];

}

return toCrypt;

}

template<typename Type>

extern void showList(list<Type> toShow);

}

Crypt.h

#pragma once

#include <iostream>

#include <string>

#include <map>

#include <list>

using namespace std;

namespace VideoStore

{

string enterString(string message);

int enterInt(string message);

int enterIndex(string message, int maxValue);

double enterDouble(string message);

string stringToLower(string str);

string enterPassword(string message);

**Продолжение приложения А**

string cryptor(string toCrypt);

template<typename Type>

static void showList(list<Type> toShow);

}

Customer.cpp

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <map>

#include <vector>

#include "Account.h"

using namespace std;

namespace VideoStore

{

Customer::Customer()

{

\_type = "customer";

name = "DefaultCustomer";

\_password = cryptor("1");

\_money = 0;

}

Account\* Customer::signUp()

{

Customer\* customer = dynamic\_cast<Customer\*>(Account::signUp());

if (customer)

{

ofstream fout("Customers.txt", ios\_base::app);

fout << \*customer;

}

return customer;

}

void Customer::buy()

{

while (!\_crate.empty())

{

\_purchased.push\_back(\_crate.back());

\_crate.pop\_back();

}

\_money -= totalPrice;

}

void Customer::info()

**Продолжение приложения А**

{

cout << "\tИнформация об аккаунте" << endl;

cout << "Ваш текущий статус: покупатель. Вы можете просматривать каталог видосов, делать поисковые запросы, покупки" << endl;

cout << "\nИмя пользователя: " << name << ", количество денег: " << \_money << '$' << endl;

cout << "\tВидосики в корзине:" << endl;

if (\_crate.empty())

{

cout << "Отсутствуют" << endl;

}

else

{

for (unsigned int i = 0; i < \_crate.size(); i++)

{

cout << i + 1 << ". " << \*\_crate.at(i) << endl;

}

}

cout << "\n\tКупленные видосов:" << endl;

if (\_purchased.empty())

{

cout << "Отсутствуют" << endl;

}

else

{

for (unsigned int i = 0; i < \_purchased.size(); i++)

{

cout << i + 1 << ". " << \*\_purchased.at(i) << endl;

}

}

}

}

IO.cpp

#include "Videoo.h"

#include "Account.h"

using namespace std;

namespace VideoStore

{

ostream& operator<< (ostream& out, Video& video)

**Продолжение приложения А**

{

out << video.\_author << ", \"" << video.\_name << "\" (";

for (auto genre = video.\_genre.begin(); genre != video.\_genre.end() - 1; genre++)

{

out << \*genre << ", ";

}

out << \*(video.\_genre.end() - 1) << ") за " << video.\_price << "$.";

return out;

}

ofstream& operator<< (ofstream& out, Video& video)

{

out << video.\_author << '|' << video.\_name << '|';

for (string str : video.\_genre)

{

out << str << ',';

}

out << '|' << video.\_price << '|';

return out;

}

ifstream& operator>> (ifstream& fin, Video& video)

{

try

{

getline(fin, video.\_author, '|');

getline(fin, video.\_name, '|');

string buff;

if (fin.peek() != ',')

{

video.\_genre.clear();

do

{

getline(fin, buff, ',');

video.\_genre.push\_back(buff);

} while (fin.peek() != '|' && fin);

fin.ignore(1);

}

else

{

fin.ignore(2);

}

getline(fin, buff, '|');

if (!buff.empty())

{

video.\_price = stod(buff);

}

**Продолжение приложения А**

else

{

video.\_price = 0;

}

}

catch (...)

{

delete& video;

throw;

}

return fin;

}

ifstream& operator>> (ifstream& fin, Admin& admin)

{

try

{

getline(fin, admin.name, '|');

getline(fin, admin.\_password, '\n');

}

catch (...)

{

delete& admin;

}

return fin;

}

ofstream& operator<< (ofstream& out, Customer& customer)

{

out << customer.name << '|' << customer.\_password << '|' << customer.\_money << '|';

if (customer.\_purchased.empty())

{

out << ",|\n";

return out;

}

else

{

unsigned int i = 0;

for (i; i < customer.\_purchased.size() - 1; i++)

{

out << \*customer.\_purchased.at(i) << ',';

}

out << \*customer.\_purchased.at(i) << "|\n";

}

return out;

**Продолжение приложения А**

}

ifstream& operator>> (ifstream& fin, Customer& customer)

{

try

{

getline(fin, customer.name, '|');

getline(fin, customer.\_password, '|');

string buff;

getline(fin, buff, '|');

if (!buff.empty())

{

customer.\_money = stod(buff);

}

else

{

customer.\_money = 0;

}

if (fin.peek() != ',')

{

do

{

Video\* video = new Video();

fin >> \*video;

customer.\_purchased.push\_back(video);

} while (fin.peek() != '|' && fin);

}

getline(fin, buff, '\n');

}

catch (...)

{

delete& customer;

throw;

}

return fin;

}

}

Main.cpp

#include <iostream>

#include <conio.h>

#include <string>

#include "Videoo.h"

#include "Account.h"

#include "Crypt.h"

#include "Store.h"

**Продолжение приложения А**

using namespace std;

using namespace VideoStore;

namespace VideoStore

{

void load(Store& store);

void save(Store& store);

bool accountExists(string name);

void addAccount(Account\* account);

map<string, Account\*> \_users = map<string, Account\*>();

}

int main()

{

setlocale(LC\_ALL, "Russian");

Store store;

load(store);

Account::setStore(&store);

unique\_ptr<Account\*> account;

cout << "1. - Зарегистрироваться\n2. - Войти в аккаунт\n3. - Войти как гость\nEsc. - Выйти";

while (true)

{

account = 0;

char ch = \_getch();

switch (ch)

{

case '1':

{

system("cls");

cout << "1. - Зарегистрироваться как покупатель\n2. - Зарегистрироваться как администратор\nEsc. - Вернуться в прошлое меню\n";

while (true)

{

ch = \_getch();

switch (ch)

{

case '1': case '2':

{

system("cls");

cout << "\tРегистрация пользователя" << endl;

delete account.release();

**Продолжение приложения А**

if (ch == '1')

{

account = make\_unique<Account\*>(new Customer());

}

else

{

account = make\_unique<Account\*>(new Admin());

}

account = make\_unique<Account\*>(move(\*account)->signUp());

if (\*account)

{

cout << "\nРегистрация прошла успешно" << endl;

system("pause");

(\*account)->menu();

}

system("pause");

delete account.release();

break;

}

case 27:

break;

default:

continue;

}

break;

}

break;

}

}

namespace VideoStore

{

void load(Store& store)

{

ifstream fin;

fin.open("Admins.txt");

while (fin && !fin.eof())

{

try

{

Admin\* admin = new Admin();

fin >> \*admin;

addAccount(admin);

if (fin.peek() == ' ')

{

break;

**Продолжение приложения А**

}

}

catch (...)

{

continue;

}

}

fin.close();

fin.open("Customers.txt");

while (fin && !fin.eof())

{

try

{

if (!fin)

{

break;

}

Customer\* customer = new Customer();

fin >> \*customer;

if (!customer->\_purchased.empty())

{

auto& purchased = customer->\_purchased;

for (unsigned int i = 0; i < purchased.size(); i++)

{

auto videoInStore = store.findVideo(purchased.at(i));

if (videoInStore)

{

purchased.at(i) = videoInStore->video;

}

}

}

addAccount(customer);

if (fin.peek() == ' ')

{

break;

}

}

catch (...)

{

continue;

}

}

return;

}

**Продолжение приложения А**

Store.cpp

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <list>

#include <vector>

#include "Store.h"

using namespace std;

namespace VideoStore

{

void Store::addVideo()

{

Video\* video = new Video();

video->setAuthor();

video->setName();

video->setGenre();

video->setPrice();

int quantity = enterInt("Введите количество видосов, которое можно будет купить: ");

\_videoo.push\_back(new VideoStatus(video, quantity));

cout << "Видео успешно добавлено" << endl;

\_availableVideoo++;

}

void Store::deleteVideo(int index)

{

auto video = \_videoo.begin();

advanceVideo(video, index);

cout << "Видео (" << \*(\*video)->video << ") успешно удалено";

(\*video)->isAvailable = false;

\_availableVideoo--;

}

list<VideoStatus\*> Store::find(string request)

{

list<VideoStatus\*> requested;

request = stringToLower(request);

for (auto videoIterator = \_videoo.begin(); videoIterator != \_videoo.end(); videoIterator++)

{

Video video = \*(\*videoIterator)->video;

**Продолжение приложения А**

if (stringToLower(video.author()).compare(request) == 0 || stringToLower(video.name()).compare(request) == 0 || video.hasGenre(request))

{

requested.push\_back(\*videoIterator);

}

}

return requested;

}

VideoStatus\* Store::findVideo(Video\* video)

{

for (auto videoIterator = \_videoo.begin(); videoIterator != \_videoo.end(); videoIterator++)

{

if ((\*videoIterator)->video == video)

{

return \*videoIterator;

}

}

return nullptr;

}

}

Store.h

#pragma once

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <list>

#include <vector>

#include "Videoo.h"

using namespace std;

namespace VideoStore

{

template<typename Type>

void showList(list<Type> toShow);

struct VideoStatus

{

Video\* video;

int quantity;

bool isAvailable;

VideoStatus();

VideoStatus(const VideoStatus& other);

**Продолжение приложения А**

VideoStatus(Video\* video, int quantity);

friend ostream& operator<< (ostream& out, VideoStatus& video);

friend ofstream& operator<< (ofstream& out, VideoStatus& video);

friend ifstream& operator>> (ifstream& in, VideoStatus& video);

};

class Store

{

list<VideoStatus\*> \_videoo;

int \_availableVideoo;

void advanceVideo(list<VideoStatus\*>::iterator& iterator, int index);

public:

Store();

Store(const Store& other);

VideoStatus\* video(int index);

void showVideo();

void addVideo();

void deleteVideo(int index);

int videoQuantity();

Video\* addToCrate(int index);

void returnVideo(Video\* video);

list<VideoStatus\*> find(string request);

void sort();

VideoStatus\* findVideo(Video\* video);

friend void save(Store& store);

};

}

Videoo.cpp

#pragma once

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <map>

#include <vector>

**Продолжение приложения А**

#include "Crypt.h"

#include "Videoo.h"

using namespace std;

namespace VideoStore

{

Video::Video()

{

\_author = "DefaultAuthor";

\_name = "DefaultName";

\_genre.push\_back("defaultGenre");

\_price = 0;

}

Video::Video(const Video& other)

{

\_author = other.\_author;

\_name = other.\_name;

\_genre = other.\_genre;

\_price = other.\_price;

}

bool Video::operator< (Video\* other)

{

if (\_author < other->\_author)

{

return true;

}

if (\_name < other->\_name)

{

return true;

}

return false;

}

bool Video::operator> (Video\* other)

{

if (\_author > other->\_author)

{

return true;

}

if (\_name > other->\_name)

{

return true;

}

return false;

}

bool Video::operator== (Video\* other)

{

**Продолжение приложения А**

if (\_author == other->\_author && \_name == other->\_name)

{

return true;

}

return false;

}

}

Videoo.h

#pragma once

#include <iostream>

#include <fstream>

#include <conio.h>

#include <string>

#include <map>

#include <vector>

#include "Crypt.h"

using namespace std;

namespace VideoStore

{

class Video

{

string \_author;

string \_name;

vector<string> \_genre;

double \_price;

public:

Video();

Video(const Video& other);

void setAuthor();

void setAuthor(string author);

void setName();

void setName(string name);

void setGenre();

void setGenre(string genre);

void setPrice();

void setPrice(double price);

string author();

string name();

vector<string>& genre();

string genre(int genreNumber);

double price();

bool hasGenre(string genreToCompare);

friend ostream& operator<< (ostream& out, Video& video);

friend ofstream& operator<< (ofstream& out, Video& video);

friend ifstream& operator>> (ifstream& fin, Video& video);

bool operator< (Video\* other);

bool operator> (Video\* other);

bool operator== (Video\* other);

};

}

**ПРИЛОЖЕНИЕ Б**

**Диаграмма классов**

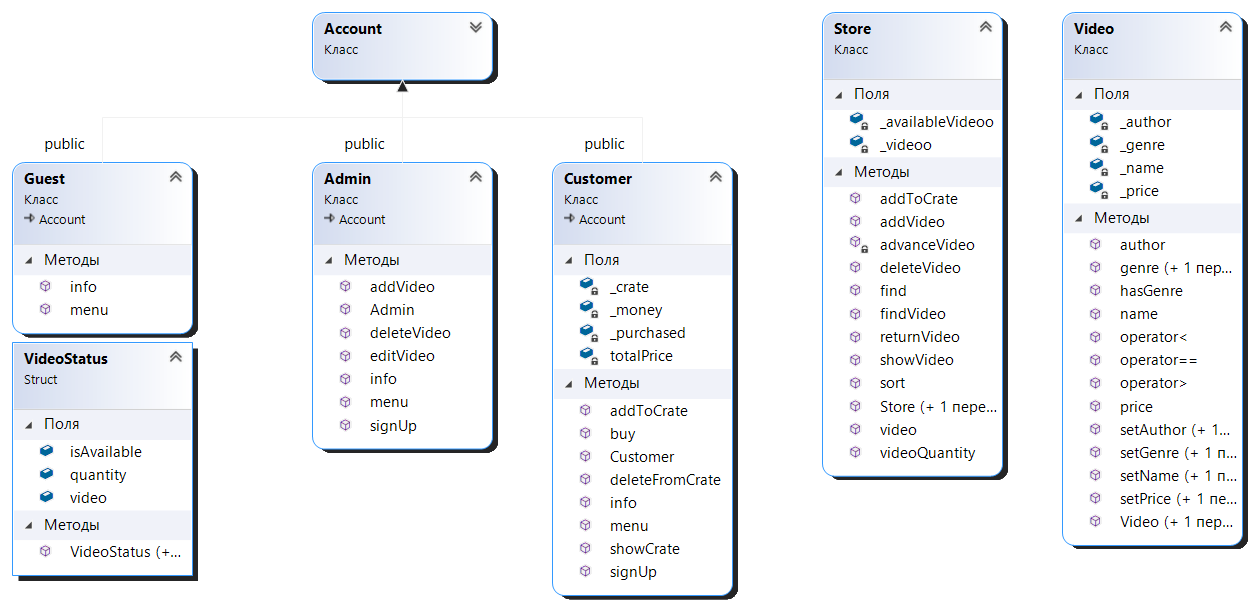


Рисунок Б.1 – Диаграмма классов

**ПРИЛОЖЕНИЕ В**

**Диаграмма вариантов использования**

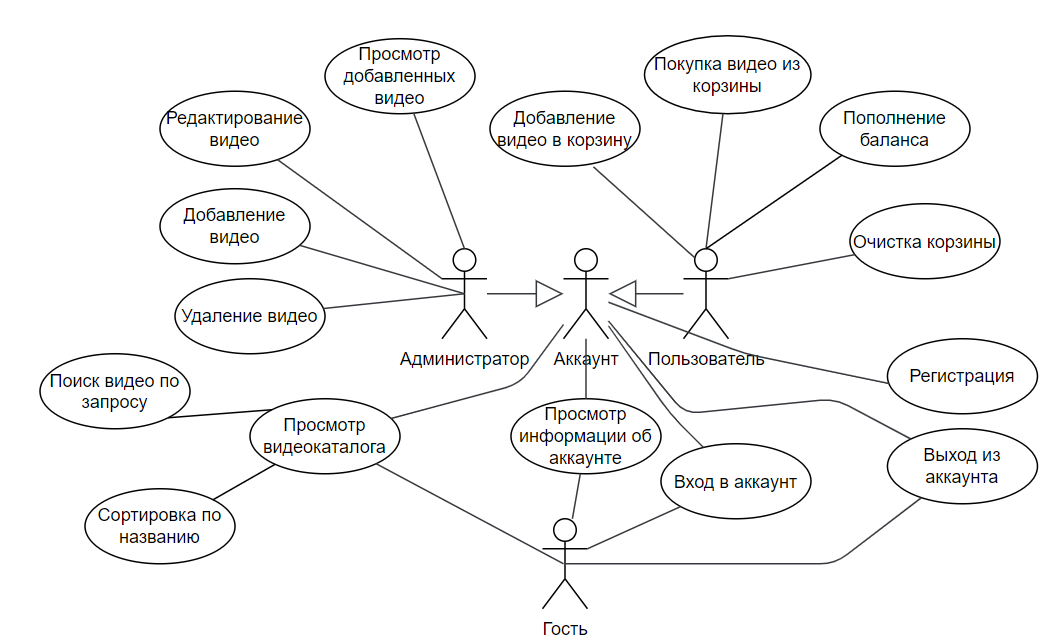
****

Рисунок В.1 – Диаграмма вариантов использования

**ПРИЛОЖЕНИЕ Г**

**Схемы алгоритмов**

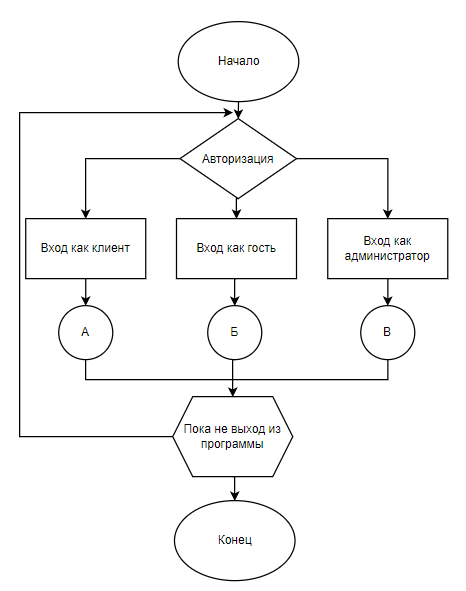


Рисунок Г.1 – Общая схема алгоритма

**Продолжение приложения Г**

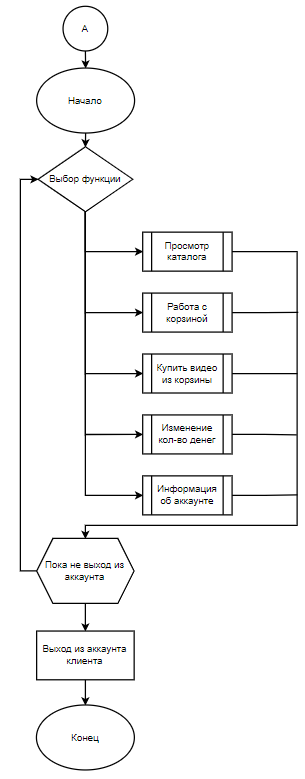


Рисунок Г.2 – Общая схема алгоритма

**Продолжение приложения Г**

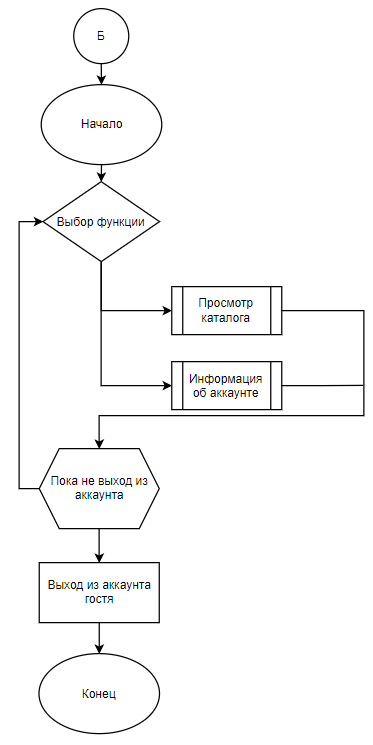


Рисунок Г.3 – Общая схема алгоритма

**Продолжение приложения Г**

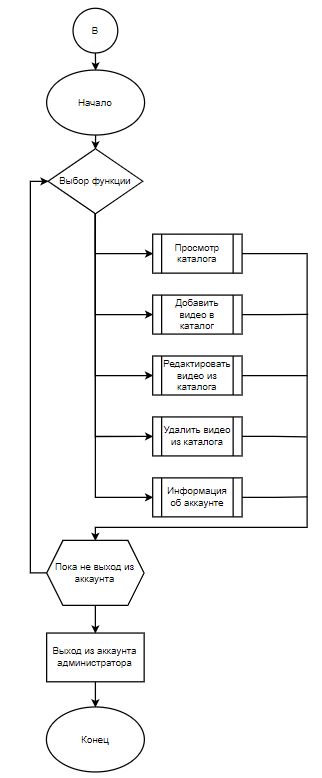


Рисунок Г.4 – Общая схема алгоритма

**Продолжение приложения Г**

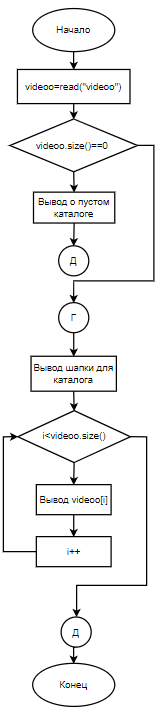


Рисунок Г.5 – Блок-схема вывод меню в консоль