Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина «Технологии разработки программного обеспечения»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  Ассистент кафедры ЭИ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.В. Федюкович |
|  | \_\_\_.\_\_\_\_.2022 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему:

**«Разработка программы учета ПРОДАЖ ТЕАТРАЛЬНЫХ БИЛЕТОВ»**

БГУИР КР 1-40 05 01-02 001 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 124402  МАЛОФЕЙ Даниил Витальевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовая работа представлена на проверку \_\_\_\_\_\_\_.2022  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2022

**РЕФЕРАТ**

БГУИР КР 1-40 05 01-02 001 ПЗ

**Малофей Д.В.** Разработка программы учета продаж театральных билетов / Д.В. Малофей. – Минск: БГУИР, 2022. – 39 с.

Пояснительная записка 39 с., 30 рис., 0 табл., 8 источников, 2 приложения

УЧЁТ ПРОДАЖИ БИЛЕТОВ, УПРАВЛЕНИЕ сеансами, МОНИТОРИНГ ДОСТУПНЫХ билетов, АНАЛИЗ популярности тивтров и спектаклей, модели *UML*, *IDEF*0, *BPMN*, схемы алгоритмов, программное средство

*Цель* *курсовой работы*: повышение качества работ касс театра, а так же оптимизация учета сеансов в кассах и ускорению взаимодействия на основе программной поддержки, учета и анализа базы в кассах театров, а также создание рейтинга популярности театров и спектаклей.

*Методология проведения работы*: в процессе решения поставленных задач использованы принципы системного подхода, аналитические методы, методы компьютерной обработки экспериментальных данных и компьютерного моделирования.

*Результаты работы*: выполнена постановка задачи и определены основные методы ее решения; в ходе объектного моделирования системы построен ряд *UML*-диаграмм; разработаны модели бизнес-процессов предметной области на основе нотаций IDEF0 и BPMN; описаны основные алгоритмы работы программного средства; разработано руководство пользователя; выполнено тестирование программного средства, показавшее его соответствие функциональным требованиям, поставленным в задании на разработку.

Программный продукт разработан на языке *C* с применением *MS Visual Studio 2019*.

*Область применения результатов*: c помощью разработанного программного средства можно проанализировать базу данных театров.

Разработанное программное средство полностью отвечает всем функциональным требованиям, необходимым при планировании, учете и анализе данных в базе данных.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc104946085)

[1 Анализ и моделирование предметной области программного средства 5](#_Toc104946086)

[1.1 Описание предметной области 5](#_Toc104946087)

[1.2 Разработка функциональной модели предметной области в нотации 5](#_Toc104946088)

[1.3 Разработка BPMN-модели основного процесса предметной области 9](#_Toc104946089)

[1.4 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 10](#_Toc104946090)

[1.5 UML - модели представления программного средства и их описание 12](#_Toc104946091)

[2 Проектирование и конструирование программного средства 17](#_Toc104946092)

[2.1 Постановка задачи 17](#_Toc104946093)

[2.2 Разработка модульной структуры программы 17](#_Toc104946094)

[2.3 Выбор способа организации данных 18](#_Toc104946095)

[2.4 Разработка перечня пользовательских функций программы 19](#_Toc104946096)

[2.5 Разработка схем алгоритмов работы программы 20](#_Toc104946097)

[3 Тестирование и проверка работоспособности программного средства 24](#_Toc104946098)

[4 Инструкция по развертыванию приложения и сквозной тестовый пример 28](#_Toc104946099)

[4.1 Авторизация 28](#_Toc104946100)

[4.2 Модуль администратора 30](#_Toc104946101)

[4.3 Модуль пользователя 31](#_Toc104946102)

[Заключение 32](#_Toc104946103)

[Список использованных источников 33](#_Toc104946104)

[ПРИЛОЖЕНИЕ А Результат “Антиплагиат” 34](#_Toc104946105)

[ПРИЛОЖЕНИЕ Б Листинг основных функций приложения 35](#_Toc104946106)

# **ВВЕДЕНИЕ**

Современные технологии и глобальная экономика ставят перед предприятиями новые вызовы, особенно в области учета импорта и экспорта продукции. В современном мире организации активно участвуют в международной торговле, что подчеркивает необходимость эффективного и надежного учета всех этапов импортных и экспортных операций. В данном контексте возникает потребность в разработке программы по учету импорта-экспорта продукции, которая способна обеспечить прозрачность, точность и оперативность ведения данных.

Импорт и экспорт продукции играют ключевую роль в развитии бизнеса, обеспечивая предприятия доступом к новым рынкам и ресурсам. Однако сложность и многосторонность процессов, связанных с таможенными операциями, документооборотом и финансовым учетом, могут представлять вызов для компаний. Разработка специализированной программы по учету импорта-экспорта становится стратегически важной для современных предприятий, стремящихся оптимизировать свою деятельность и минимизировать риски в условиях глобального бизнеса.

Целью создания данной программы является обеспечение эффективного мониторинга и управления всеми аспектами импортных и экспортных операций, начиная от заказа продукции и заканчивая ее поставкой клиенту.

Для достижения поставленной цели необходимо выполнить целый ряд задач:

* Исследовать текущие методы ведения учета импорта-экспорта продукции.
* Проанализировать существующие приложения для учета импорта-экспорта продукции.
* Определить основные слабые стороны текущего процесса учета импорта-экспорта продукции.
* Разработать функциональную модель основных процессов учета.
* Создать программную систему учета импорта-экспорта продукции.
* Провести тестирование приложения на соответствие поставленным задачам.
* Выполнить отладку и устранение ошибок, выявленных в процессе тестирования.

# **Анализ и моделирование предметной области программного средства**

### Описание предметной области

Приложение по учету импорта-экспорта продукции. В приложении есть множество функций, основными из них являются работа с продуктами и заказами.

При добавлении нового продукта, для него создается уникальный номер (id), с помощью которого с ним можно дальше взаимодействовать. Продукт в себя включает: свой уникальный номер, название продукта, цена за одну единицу, категория к которой он относиться, фабрику, которая его произвела и поставщик, который его поставил. Все эти данные дают нам полноту картины о продукте и позволяют нам с ним наиболее эффективно взаимодействовать.

При добавлении заказа ему также, как и продукту задается свой уникальный номер (id). Заказы в себя включают номер продукта, который надо поставить, номер клиента, дату поставки, а также количество единиц продукта, которое надо поставить.

### Разработка функциональной модели предметной области в нотации

Одна из функций программы – создание учетной записи. Аккаунт обычного пользователя может создать любой, а учетную запись админа – только другой админ. Создание аккаунта требует ввода уникального логина и пароля, затем логин проверяется на уникальность, а пароль на корректность и если все условия выполняются, то аккаунт будет успешно создан . Процесс создание учетной записи можно увидеть на рисунке 1.1.

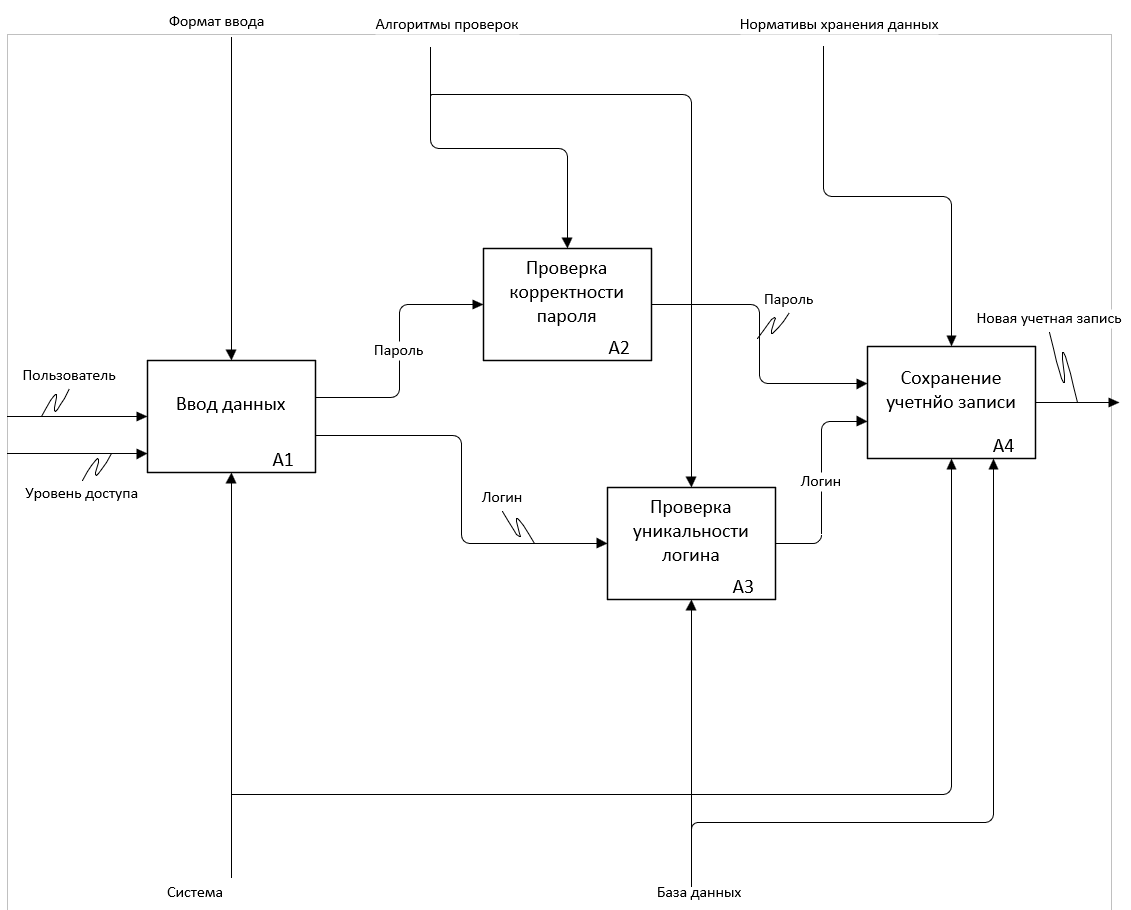


Рисунок 1.1 – Процесс создания учетной записи

После входа в программу у пользователя будет возможность купить билет. Покупка билетов подразумевает, что пользователь выбирает сеанс, а из него уже не купленные билеты. Процесс покупки билетов можно увидеть на рисунке 1.2.

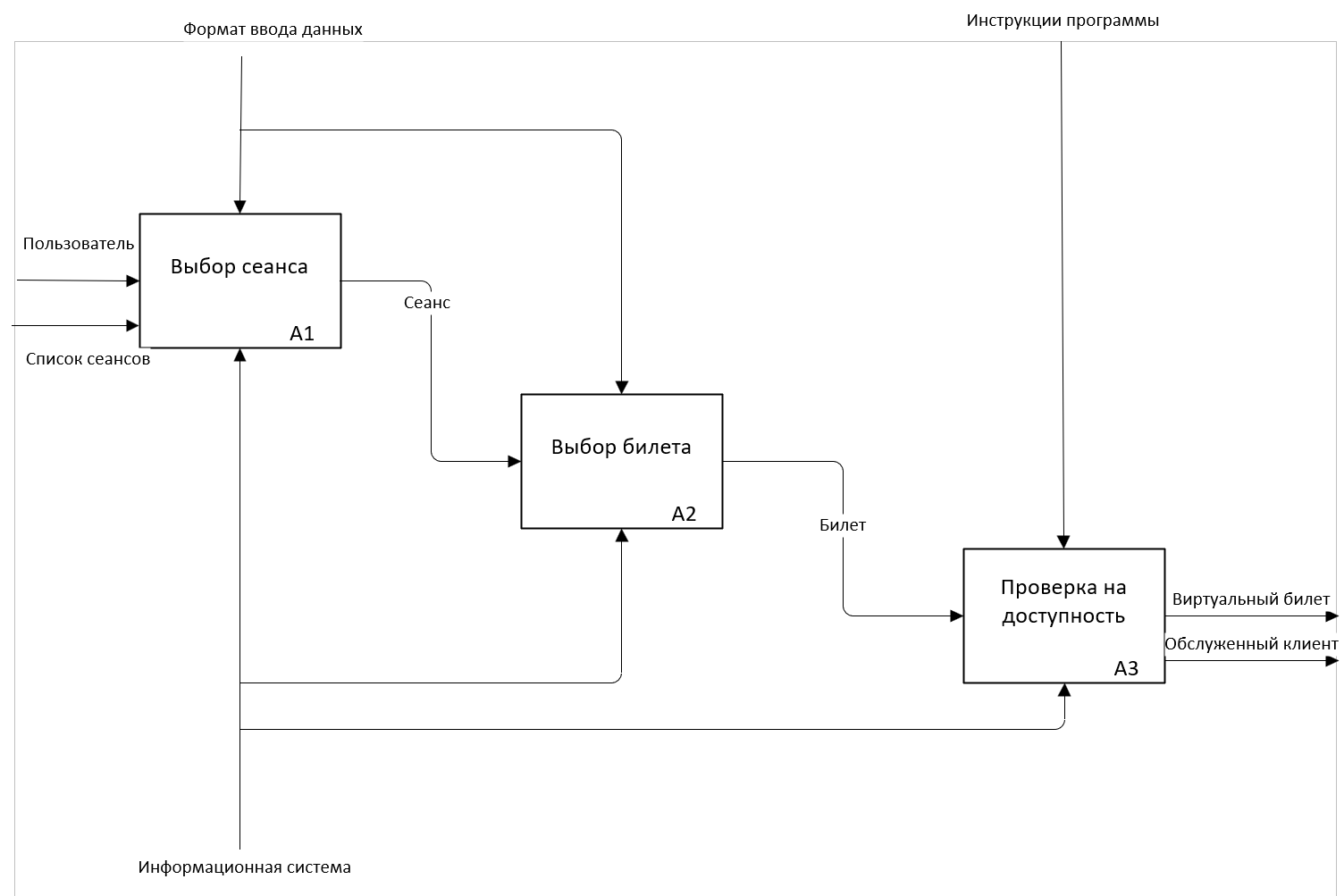


Рисунок 1.2 – Покупка билетов

После успешной покупки билета пользователь если у него нет принтера или он не хочет печатать, может распечатать свои билеты в театре. Для этого ему надо в одном из терминалов театра войти в свой аккаунт и выбрать свои билеты для печати. Общий вид взаимодействия пользователя с терминалом можно увидеть на рисунке 1.3.

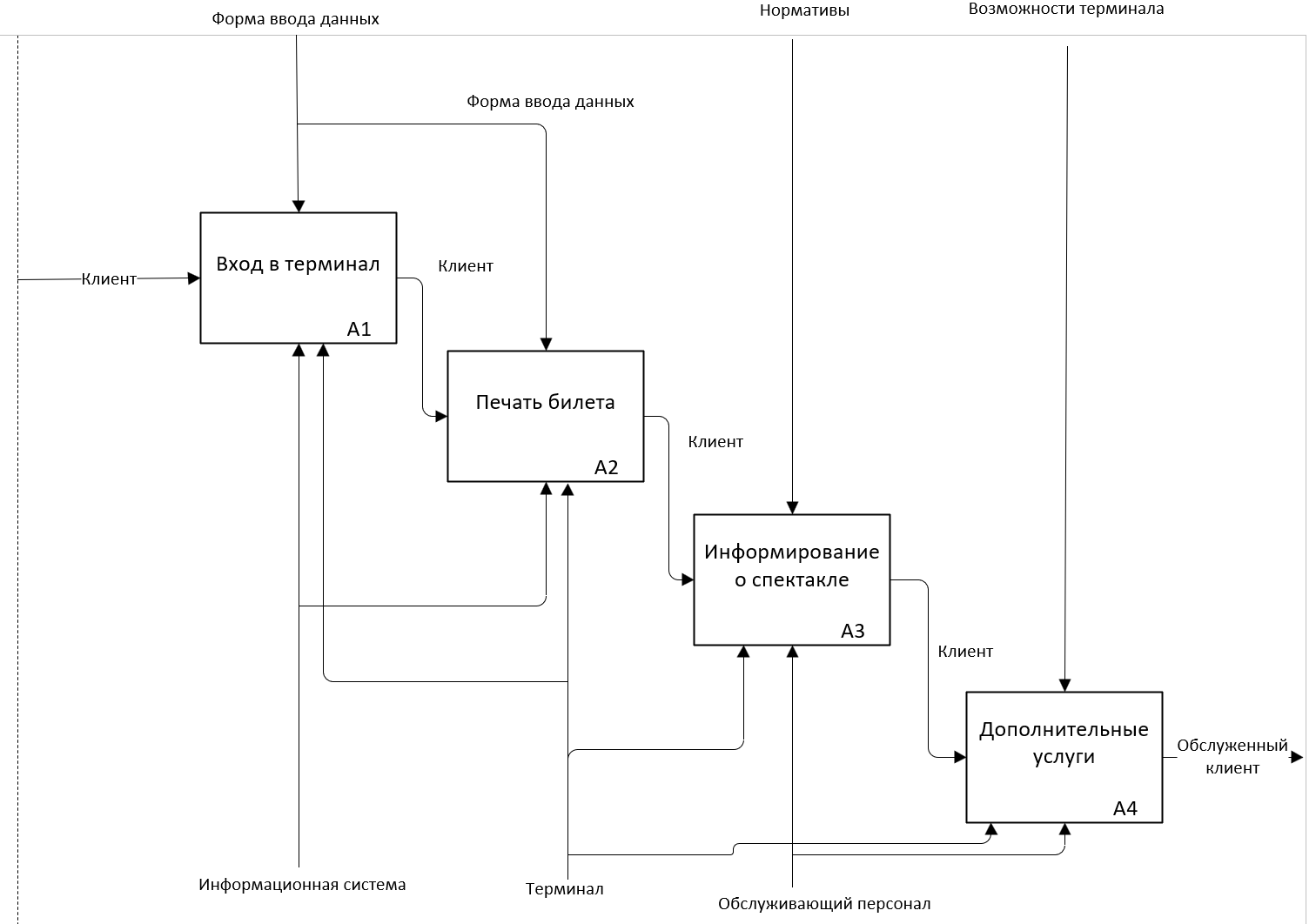


Рисунок 1.3 – Процесс взаимодействия с терминалом

Для того, чтобы театры успешно функционировали им нужны работники. Для приема на работу нужно подать резюме и пройти собеседование. Общий вид процесса приема на работу представлен на рисунке 1.4.

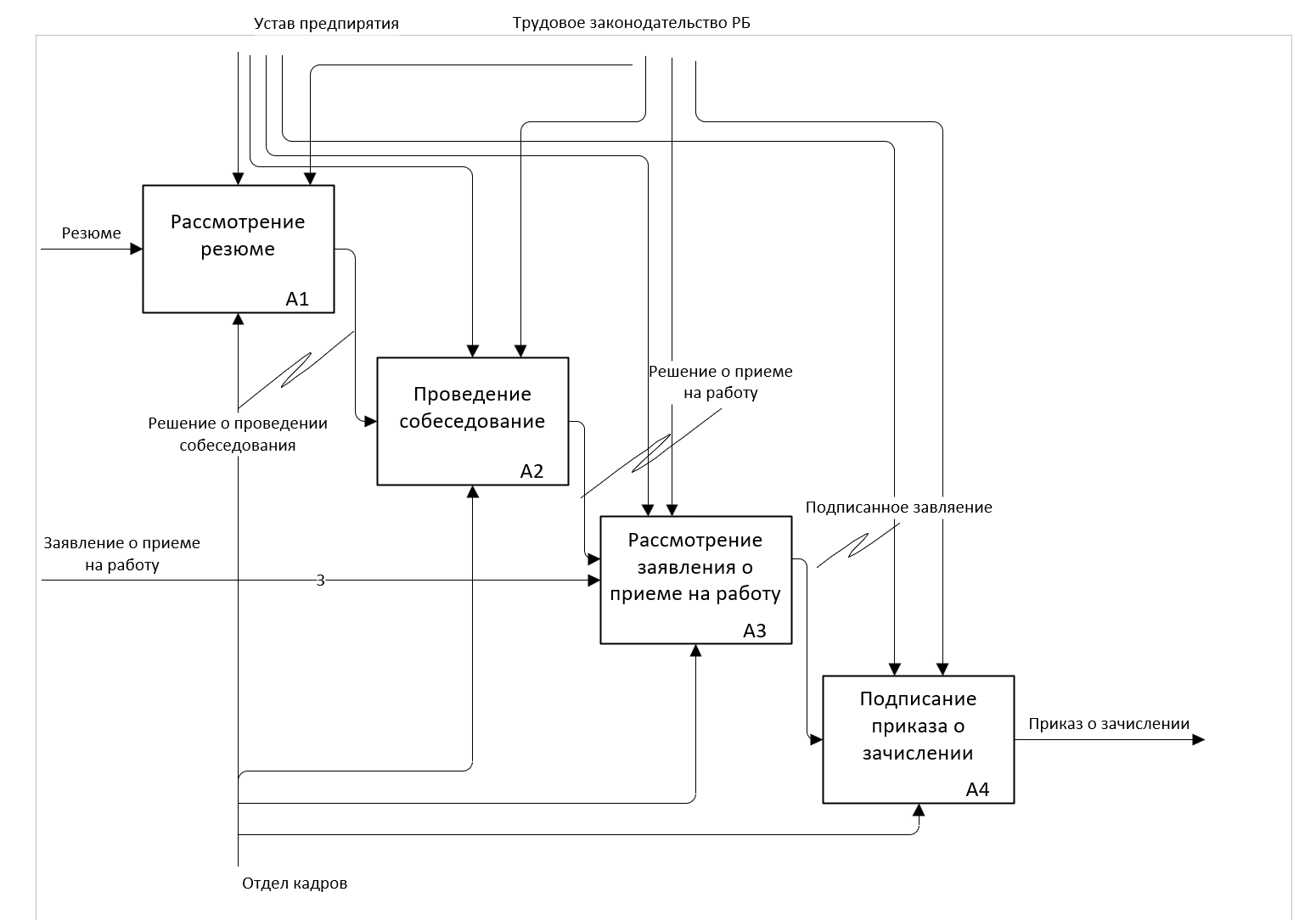


Рисунок 1.4 – Процесс приема человека на работу

### Разработка BPMN-модели основного процесса предметной области

Основным процессом программы является покупка билетов. Что бы купить билет пользователь должен сначала зайти в свой аккаунт, просмотреть сеансы и из них выбрать тот на который он хочет пойте. После ввести количество билетов, которое он хочет купить, если такое количество доступно, то пользователь выбирает и покупает билеты, в противном случае система сообщает об ошибке и предлагает пользователю выбрать другой сеанс или отменить действие и вернуться в главное меню. Ниже на рисунке 1.4, можно увидеть подробный алгоритм этого действия с момента запуска программы.

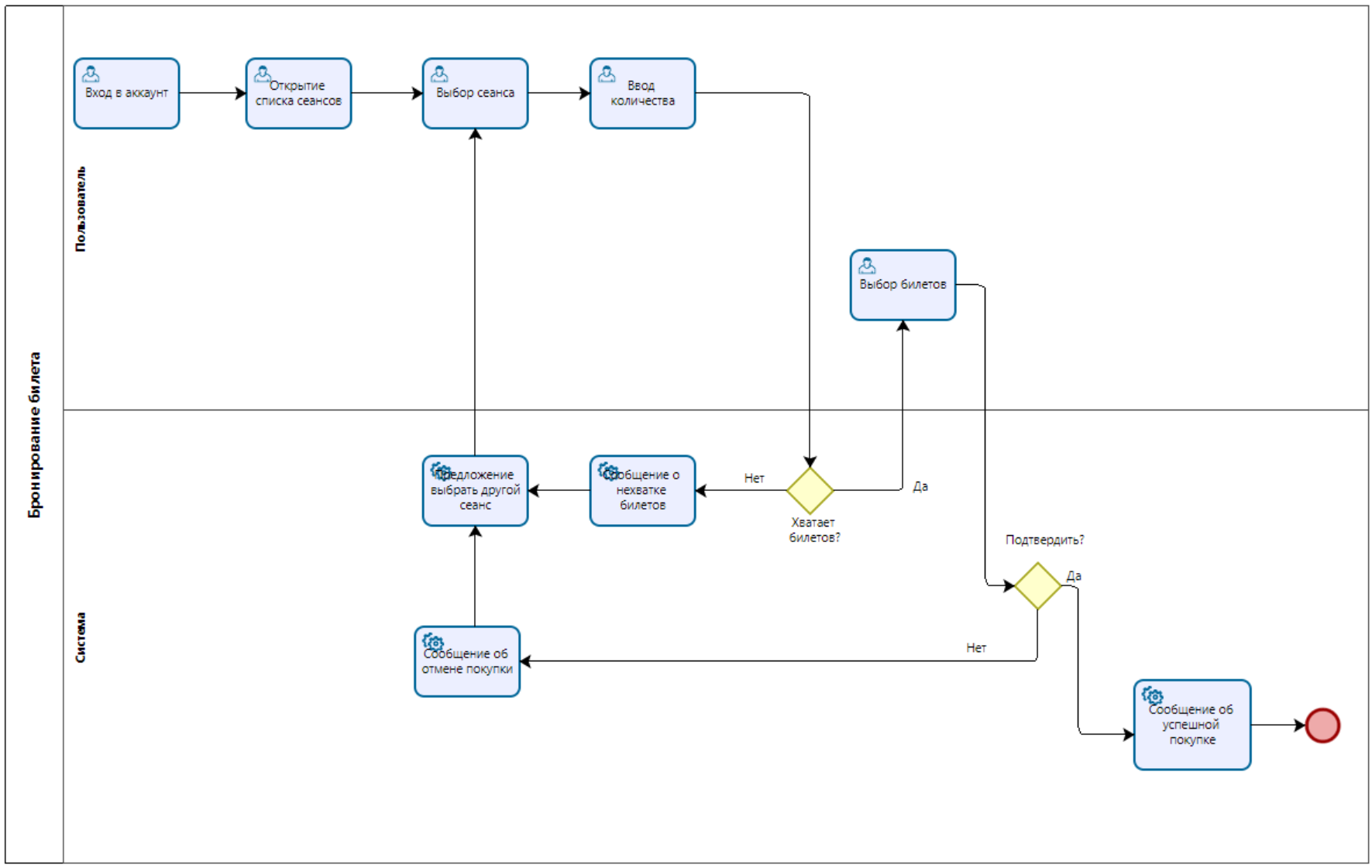


Рисунок 1.4 – Процесс покупки билетов

### Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Финальный вариант программного средства должен позволять свободно выполнять ряд требуемых функций.

Первым этапом работы программы является авторизация – предоставление прав доступа. В рамках данного этапа необходимо считать данные из файла с учетными записями пользователей следующего вида:

* login;
* password;
* role (данное поле служит для разделения в правах администраторов и пользователей);
* active (данное поле служит для заблокированния и разблокирования пользователей).

После ввода пользователем своих персональных данных (логина и пароля) и сверки со считанной из файла информацией необходимо предусмотреть возможность входа в качестве администратора (в этом случае role = 2) или пользователя (в этом случае role = 1).

Если файл с учетными записями пользователей не существует, то необходимо создать его и записать учетные данные администратора.

Вторым этапом работы программы является работа с данными, которая становится доступной только после прохождения авторизации. Данные хранятся в отдельных файлах.

Для работы с данными сеансов и билетов должны быть предусмотрены два функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора включает следующие подмодули:

1. Управление учетными записями пользователей:
   * + просмотр всех учетных записей;
     + добавление новой учетной записи;
     + редактирование учетной записи;
     + удаление учетной записи;
2. Работа с сеансами и билетами:
   1. режим редактирования:
      * + просмотр всех сеансов;
        + просмотр всех билетов
        + добавление новой сеансов и билетов;
        + удаление сеансов и билетов;
        + редактирование сеансов и билетов;
   2. режим обработки данных:
      * + выполнение индивидуального задания
        + поиск билетов;
        + сортировка сеансов.
3. Работа с файлом данных:

* запись данных в файл;
* открытие существующего файла или его создание.

Модуль пользователя включает подмодуль работы с данными со следующими функциональными возможностями:

* просмотр всех сеансов;
* просмотр всех не купленных билетов;
* покупка билетов;
* сортировка сеансов по различным признакам;
* выполнение индивидуального задания;
* поиск билета.

Для реализации перечисленных модулей/подмодулей необходимо создавать меню с соответствующими пунктами.

Предусмотреть обработку исключительных ситуаций:

* введенные пользователем данные не соответствуют формату поля *(например, символы в числовом поле);*
* введенные пользователем данные некорректны *(например, месяц слишком большой (>12) или слишком маленький (<1));*
* файл с данными для чтения не существует;
* ничего не найдено по результатам поиска;
* номер удаляемой записи выходит за пределы массива;
* логин новой учетной записи уже существует;
* возможность возврата назад (навигация);
* запрос на выполнение необратимых действий, а именно, подтверждение удаления сеанса «Вы действительно хотите удалить сеанс?»;
* обратную связь с пользователем, например, вывод сообщения об успешности создания файла/удаления записи/ и т.д.

Ограничения по реализации программного средства:

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта к курсовой работе.
2. Имена функций должны быть осмысленными и строится по принципу глагол + существительное.
3. Не допускается использование оператора прерывания goto.
4. Код не должен содержать неименованных числовых констант, неименованных строковых констант. Подобного рода информацию следует выносить в глобальные переменные с атрибутом const.
5. Код необходимо комментировать.
6. Код не должен дублироваться – для этого существуют функция или метод.
7. Одна функция решает только одну задачу. При этом внутри функции возможен вызов других функций.
8. Выполнение операций чтения/записи в файл должно быть сведено к минимуму.
9. Следует избегать глубокой вложенности условных и циклических конструкций: вложенность блоков должна быть не более трех.
10. Следует избегать длинных функций: текст функции должен умещаться на один экран (размер текста не должен превышать 25–50 строк).
11. Следует выносить код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

### UML - модели представления программного средства и их описание

С помощью UML моделей далее будут отображены некоторые аспекты приложения.

Начнем с представления функциональных возможностях пользователя. Ниже на рисунке 1.5 можно увидеть общий список функций, доступных клиентам. Они разбиты на группы взаимодействия с кассами театров и с аккаунтом. Взаимодействие с аккаунтом представляет собой вход в систему посредством ввода своего логина и пароля и созданием аккаунта, посредством ввода нового логина и дважды введения одного и того же пароля. Взаимодействие с кассами театра представлено следующими функциями: покупка билетов, просмотр всех не проданных билетов, сортировка сеансов по выбранным параметрам, поиск информации о билете по его уникальному номеру, выведению списка самых популярных театров и спектаклей в определенный промежуток времени. Ниже на рисунке 1.5 можно увидеть подробный разбор этих функций.

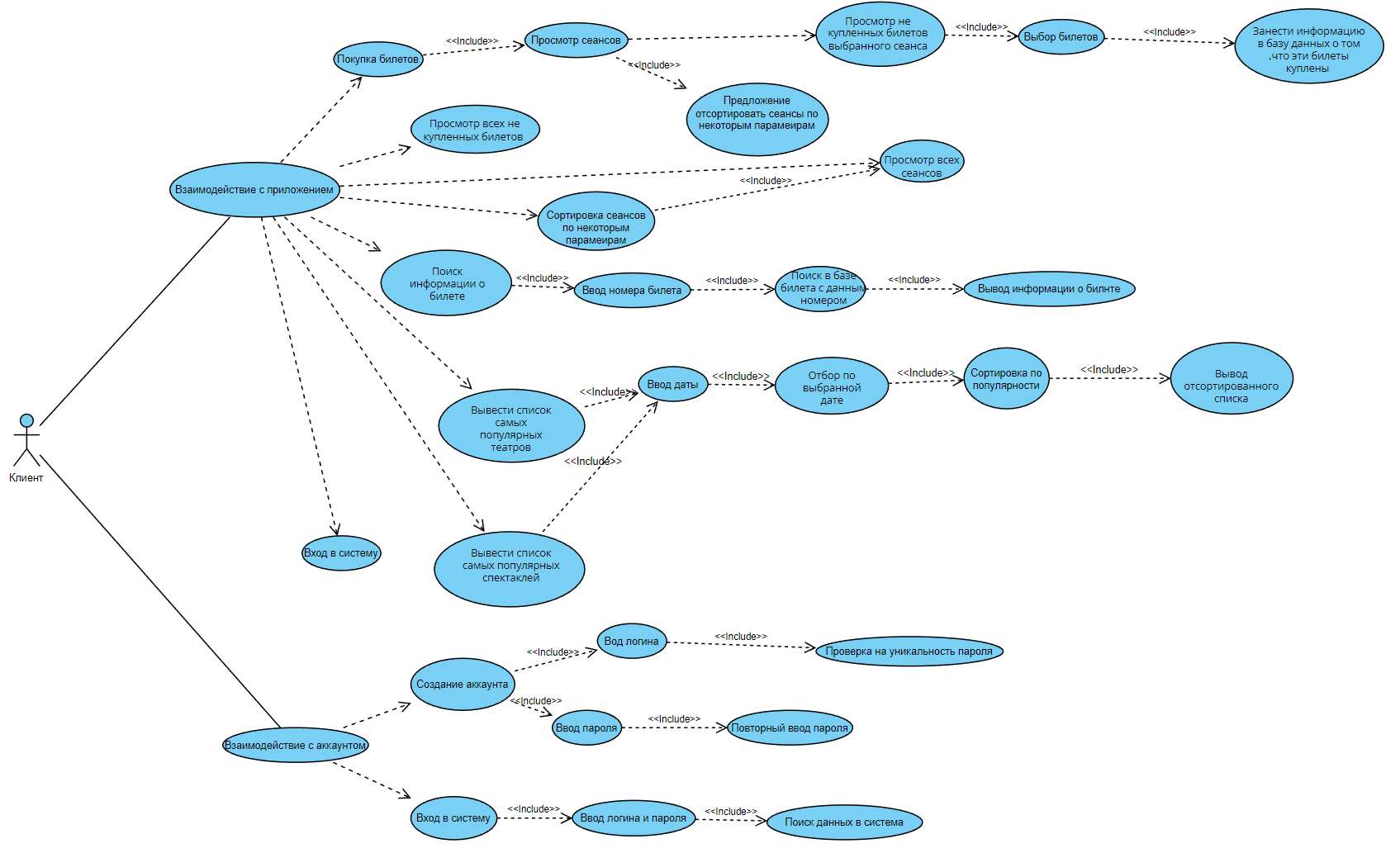


Рисунок 1.5 – Модель вариантов использования

Благодаря модели последовательности можно более подробно проследить за порядком работы программы. На ней видно, какие процессы выполняются раньше других и более подробно описаны процессы происходящие в это время. Ниже представлены процессы входа в систему, просмотр сеансов билетов, а также процесс покупки билета. Подробно эти процессы рассмотрены на рисунок 1.6.

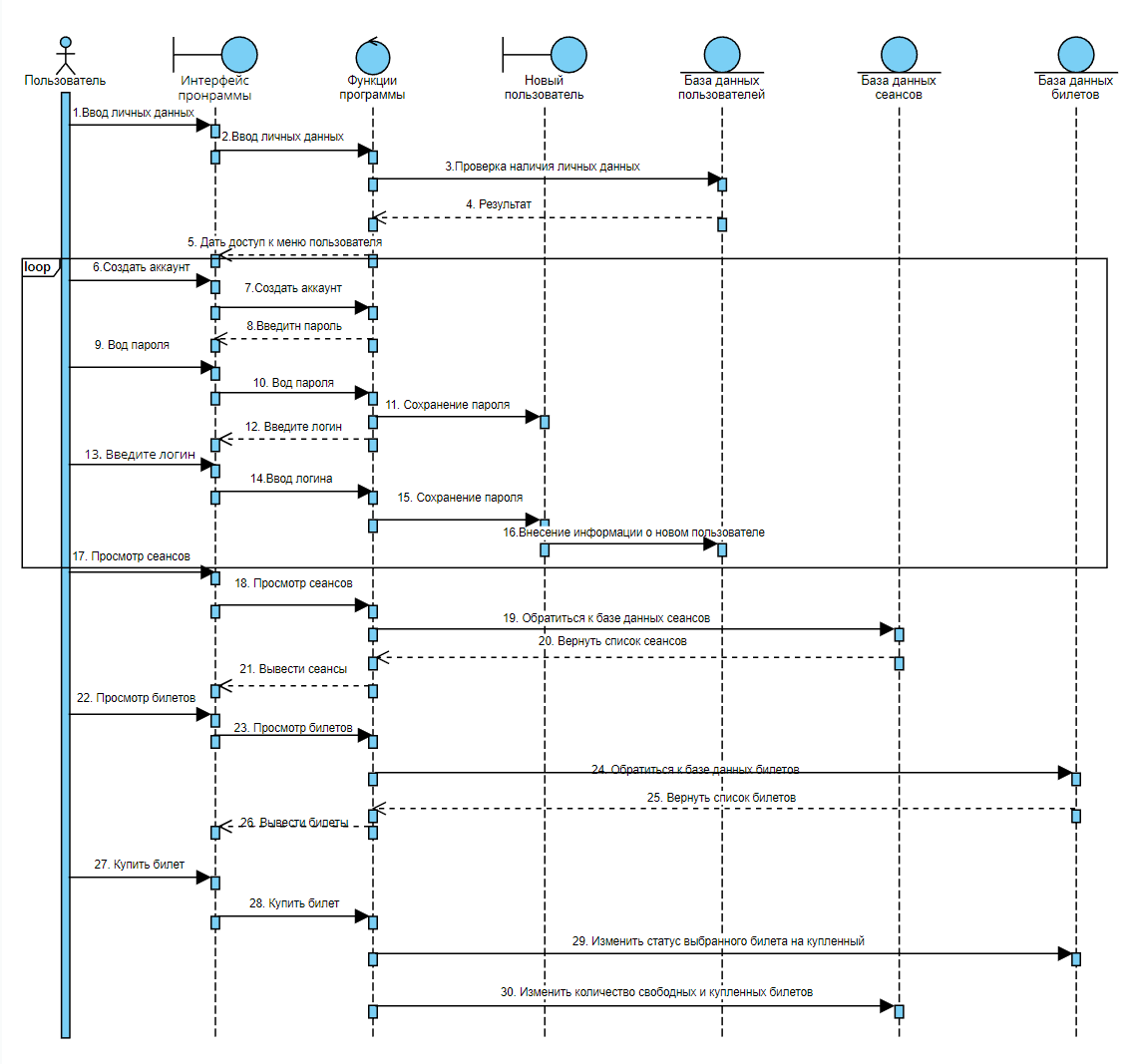


Рисунок 1.6 – Модель последовательности

Далее более подробно рассмотрим одну из главных функций взаимодействия с кассами театров: покупка билетов. Ниже на рисунке 1.6 указан процесс, который происходит при покупки билетов, со всеми надлежащими проверками и изменениями базы данных.

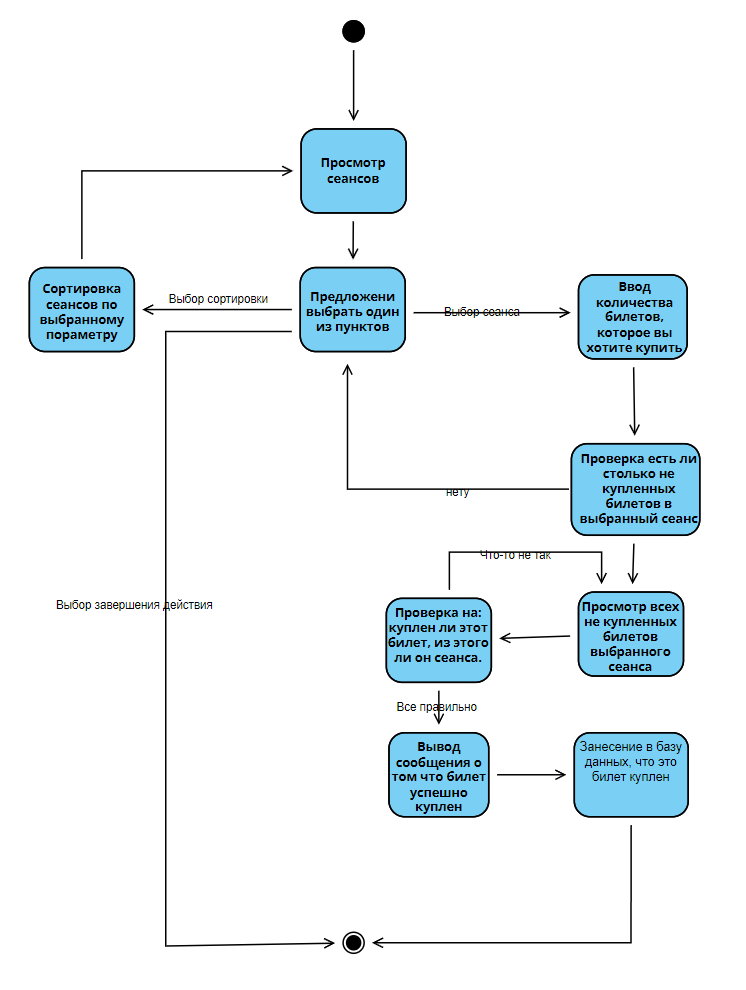


Рисунок 1.6 – Модель состояния

Далее требуется рассмотреть, где будет использоваться и поддерживаться приложение. Поскольку приложение задача, которого состоит в том, чтобы формировать рейтинг популярности театров спектаклей, то приложение не нуждается в мультиплатформенности, отсюда следующая картина. На рисунке 1.7 представлена модель компонентов, которые будут участвовать в работе программы.

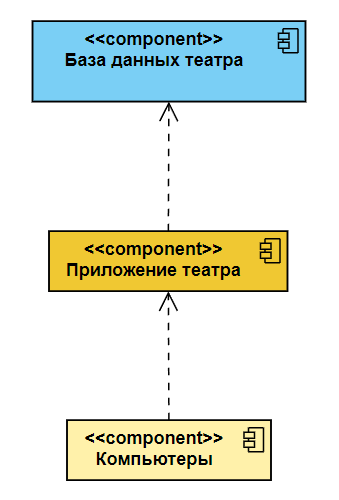


Рисунок 1.7 – Модель компонентов

И наконец-то рассмотрим модель развертывания. На ней более подробно отображается то, какими методами приложение будет представлено и связи этих методов между собой. Смотрите рисунок 1.8.

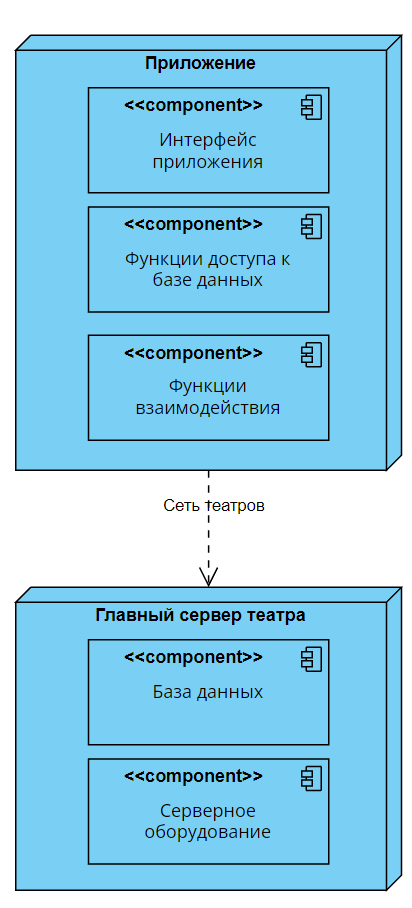


Рисунок 1.8 – Модель развертывания

# **Проектирование и конструирование программного средства**

### Постановка задачи

Основная задача – разработать программу для взаимодействия с базой данных касс театров. ПО будет поддерживать систему аккаунтов: admin, user. Каждый тип аккаунта будет иметь свой список функций и возможностей. Админ будет иметь доступ к редактированию базы данных сеансов и пользователей, регистрации новых и удалению старых аккаунтов. Обычный юзер будет иметь доступ только к функциям, позволяющим пользоваться услугами театров такими как просмотр сеансов и не купленных билетов, просмотр рейтинга популярности спектаклей и театров, покупка билетов. Программа будет реализована на языке C с элементами C++ в среде Microsoft Visual Studio.

### Разработка модульной структуры программы

У разработанного приложения для касс театров есть своя структура (см. рисунок 2.1). Основными звеньями в ней являются модули пользователя и админа. У каждого модуля есть свой перечень доступных ему функций. У пользователя есть доступ только к функциям взаимодействия с билетами и сеансами, у админа добавляются функции работы с учетными записями и расширяются функции работы с сеансами.

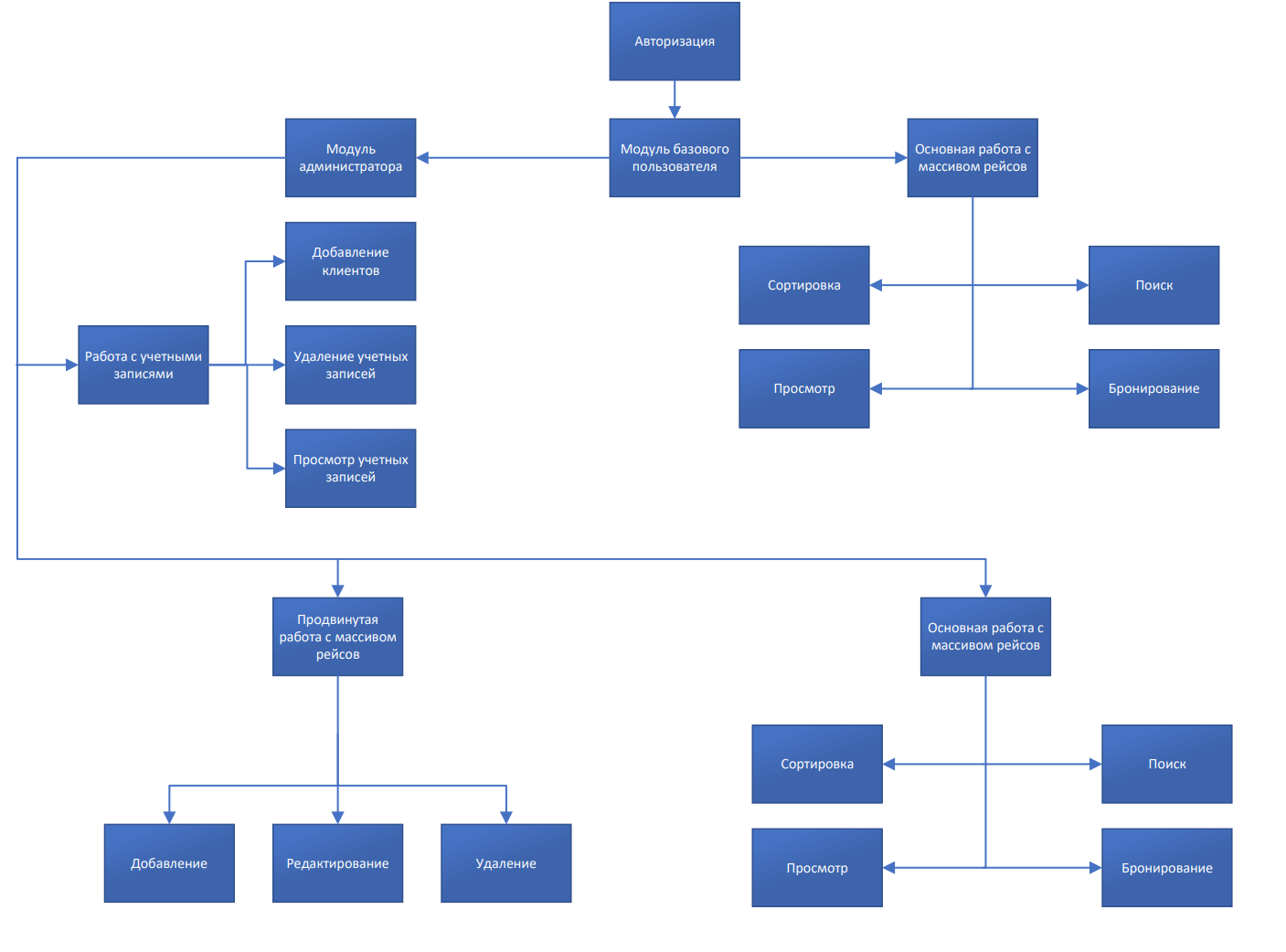


Рисунок 2.1 – Модульная структура программы

### Выбор способа организации данных

В разработанном приложении реализованы 3 класса.

Класс GeneralInformationBilets является контейнером для работ с сеансами. Он содержит поля:

* teatr;
* spectacl;
* day;
* month;
* year;
* kollBuy;
* kollOst;
* price;

Поле teatr хранит название театра, spectacl – название спектакля. Поля day, month и year хранят год месяц и число соответственно, kollBuy – количество купленных билетов, kollOst – количество оставшихся(не купленных) билетов, price – цена одного билета .

Класс Bilet является контейнером для работ с билетами. Он является наследником класса GeneralInformationBilets и содержит поля:

* teatr;
* spectacl;
* day;
* month;
* year;
* price;
* nomer;
* buy;
* row;
* place.

Поле teatr, spectacle, day, month, year, kollBuy, kollOst, price – хранят туже информацию, что и в классе GeneralInformationBilets. Поле nomer хранит уникальный номер билета, который автоматически присваивается каждому билету при создании сеанса. Поле row хранит к какому ряду принадлежит место, а place место в ряде. Поле buy хранит информацию о том куплен ли билет.

Класс User является контейнером для данных пользователя. Она содержит поля:

* login;
* password;
* active;
* role.

Поле login хранит в себе уникальное имя учетной записи пользователя, password – пароль, active – активен ли аккаунт (если 0, то не активен, если 1, то активен). Поле role отвечает за определение типа учетной записи, если status = 1, то это аккаунт обычного клиента, если status = 2, то это учетная запись админа.

### Разработка перечня пользовательских функций программы

Функции взаимодействия с билетами:

* viewBiletsForBuy – просмотр (пользователем) не купленных билетов;
* viewBilets – просмотр (админом) всех билетов;
* Buy – покупка билета;
* deleteBilet – удаление билета (администратором);
* FindBilets – поиск билета.

Функции взаимодействия с сеансами:

* addNewBilets – добавление новых сеансов;
* viewGeneralInformationBilets– просмотр всех сеансов;
* buyBilets – выбор сеанса из которого, пользователь хочет купить билет;
* deleteGeneralInformationBilets – удаление сеанса (администратором).

Функции взаимодействия с учетными записями:

* enterPassword – ввод пароля;
* Log\_in – вход в систему;
* viewUsers – просмотр пользователей (администратором);
* addition – создание нового аккаунта;
* deleteSearchUser – удаление пользователя (администратором);
* verifyUsers – разблокировка учетной записи (администратором);
* editingUsers – редактирование учетных записей пользователей (администратором);
* createNewSystem – создание самого первого аккаунта администратора.

Функции сортировки:

* SortBilets – меню для выбора сортировки;
* SortingByPrice – сортировка сеансов по цене;
* SortingByData – SortingBy\_kollOst
* SortingBy\_kollOst – сортировка сеансов по количеству оставшихся билетов;
* SortingBy\_kollBuy – сортировка сеансов по количеству купленных билетов;
* SortingByTeatr – сортировка сеансов по названию театра;
* SortingBySpectacl – сортировка сеансов по названию спектакля.

Консольные функции:

* UserFunction – доступ к функциям пользователя;
* AdminFuction – доступ к функциям админа.

### Разработка схем алгоритмов работы программы

Алгоритм функции ChangeBilets в виде схемы (см. рисунок 2.2). С начало администратору выводятся все сеансы. Он должен ввести номер сеанса, который хочет редактировать, если такого номера нет, то действие проверяется. Если всё хорошо, то ему выводится информация о данном сеансе и предлагают редактировать его поля. После редактирования, проверяется есть ли в базе данных такой же сеанс, если есть то выводится сообщение о ошибке и предлагается заново изменить поля, если совпадении не обнаружено, то информация вноситься в базу данных.

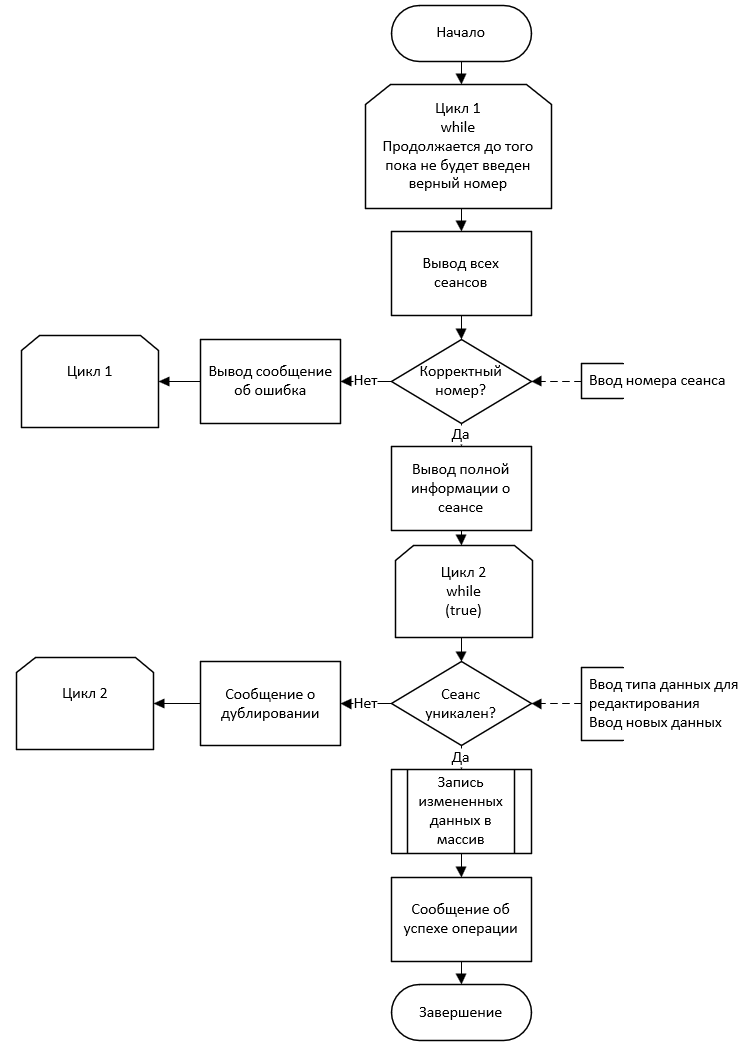


Рисунок 2.2 – Алгоритм функции ChangeBilets

Алгоритм функции buyBilets (см. рисунок 2.3). С начало пользователю выводится список сеансов и предлагается выбрать билеты какого сеанса он хочет купить. После выбора ему выводится информация только об этом сеансе и спрашивается сколько билетов он хочет купить. Если введенное им количество больше чем свободно билетов на данный сеанс, то ему предлагается выбрать другой, или завершить покупку, если все нормально, то ему выводится список свободных билетов на данный сеанс. Пользователь вводит уникальный номер билета, который он хочет купить. Если введенный им номер билета принадлежит выбранному сеансу и этот билет еще не куплен, то выводится сообщение об успешной покупки и заносятся изменения в базу данных, в противном же случае выводится ошибка и пользователю предлагают выбрать другой билет.

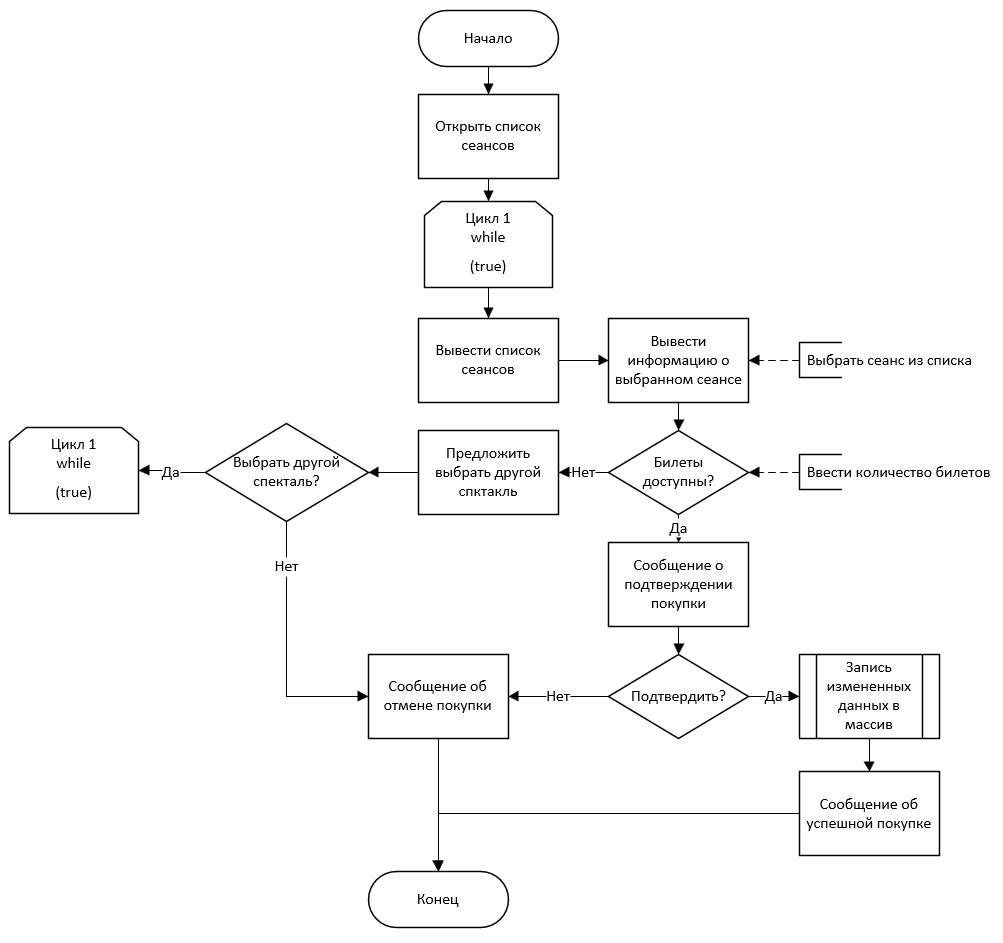


Рисунок 2.3 – Алгоритм функции buyBilets

Алгоритм функции addition (см. рисунок 2.4). При создании аккаунта требуется ввести логин и 2 раза пароль. Происходит проверка на совпадение введенных паролей, если пароли не совпадают, то выводится сообщение об ошибке и происходит завершение функции, если пароли совпадают, то происходит поиск такого же логина в базе данных. Если такой логин найден, то выводится сообщение о том, что такой логин уже существует и происходит завершение функции, если логин уникален, то новый пользователь заносится в базу данных.

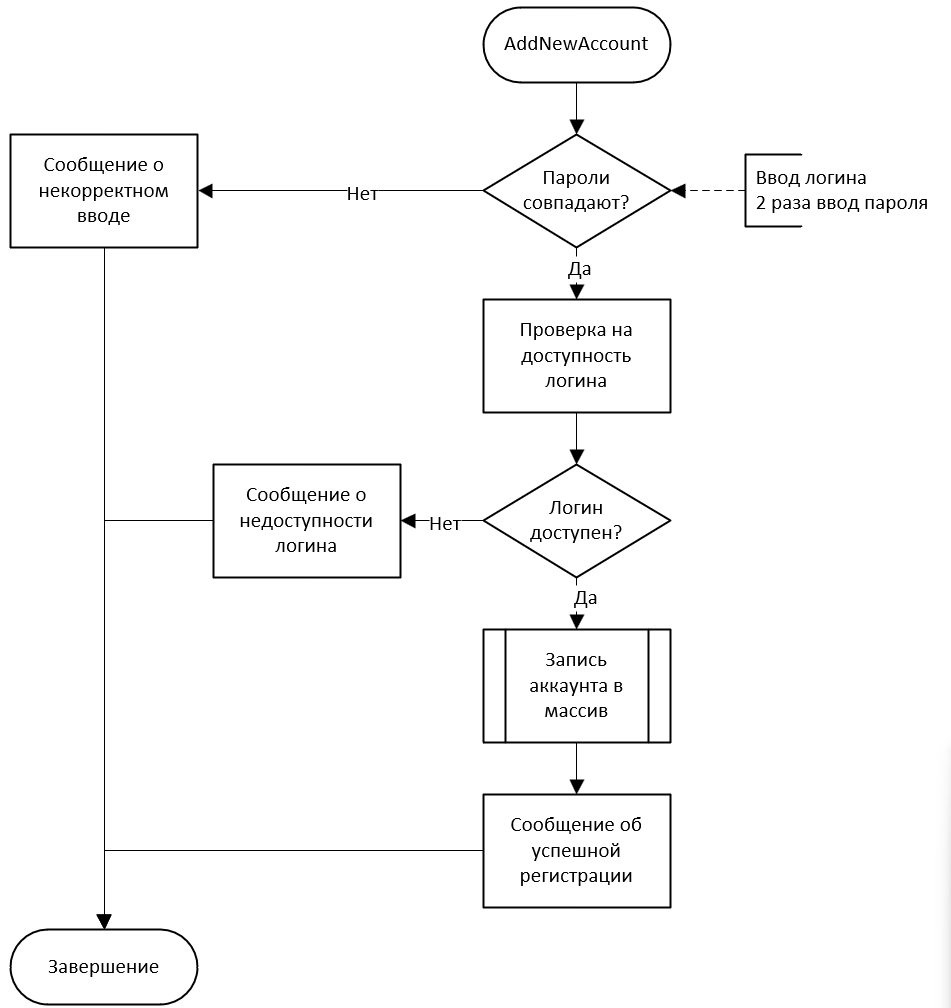


Рисунок 2.4 – Алгоритм функции addition

# **Тестирование и проверка работоспособности программного средства**

В программе учитывается возможность ошибки пользователя при работе с ней. Добавлены различного рода проверки. Одни из них – это проверки вводимого значения и на нечисловой ввод. Результаты работы с главным меню модно увидеть на рисунках 3.1, 3.2, 3.3, 3.4. На рисунках 3.5, 3.6, 3.7, 3.8 представлена работа проверок вводимых значений в меню пользователя.

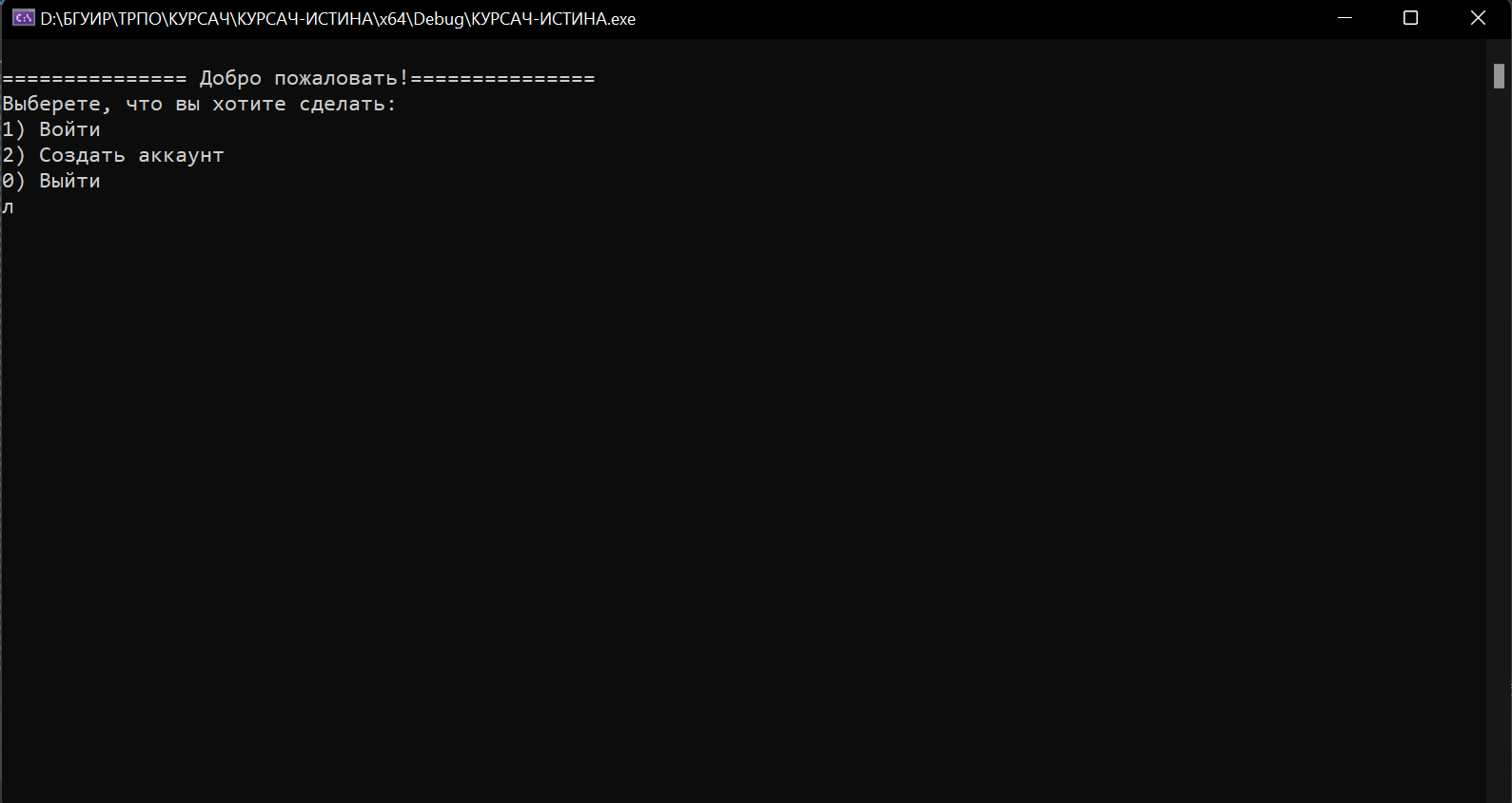


Рисунок 3.1 – Ввод нечислового значения в начальном меню

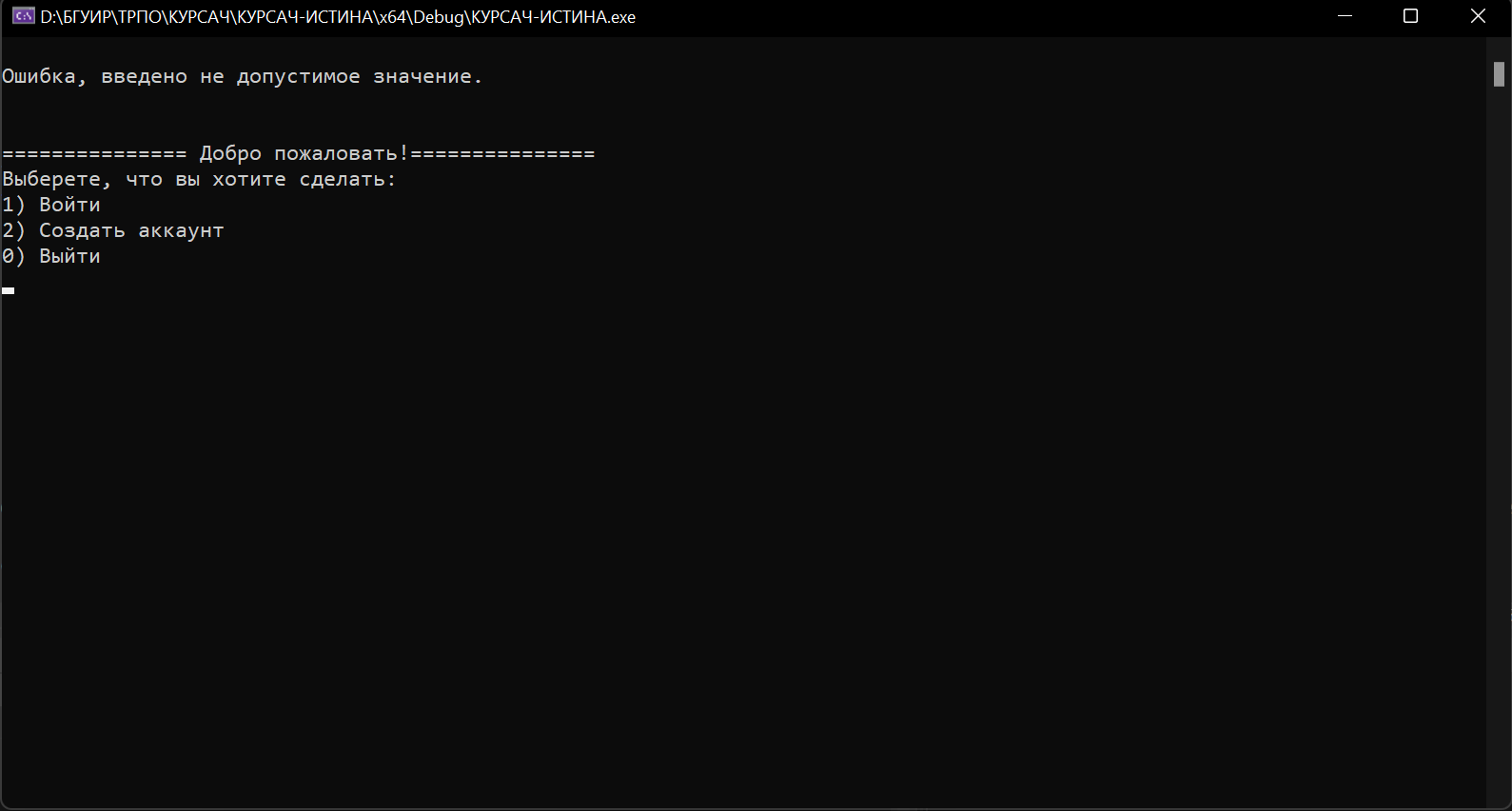


Рисунок 3.2 – Результат ввода нечислового значения в начальном меню

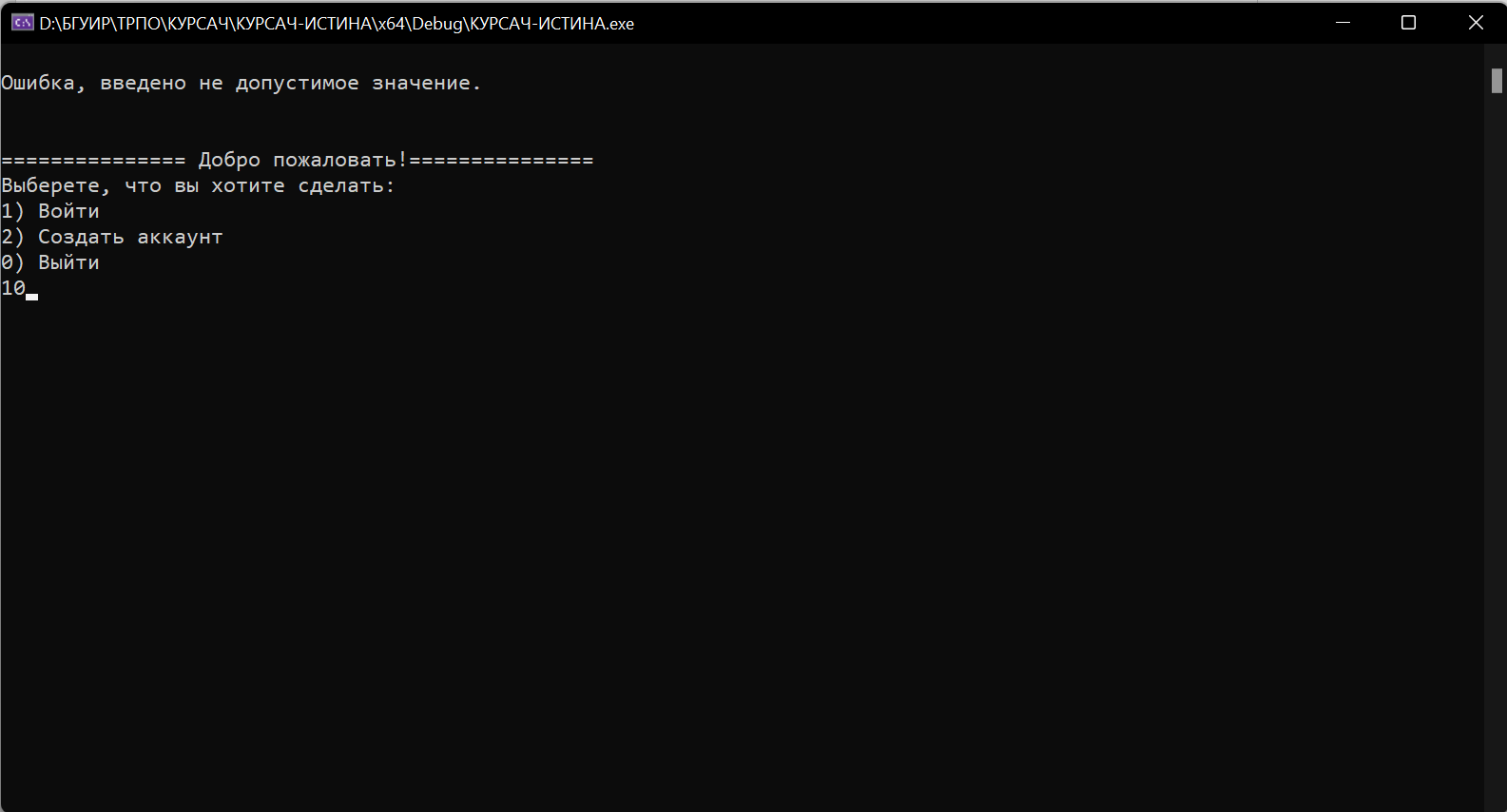


Рисунок 3.3 – Выбора пункта, которого нету

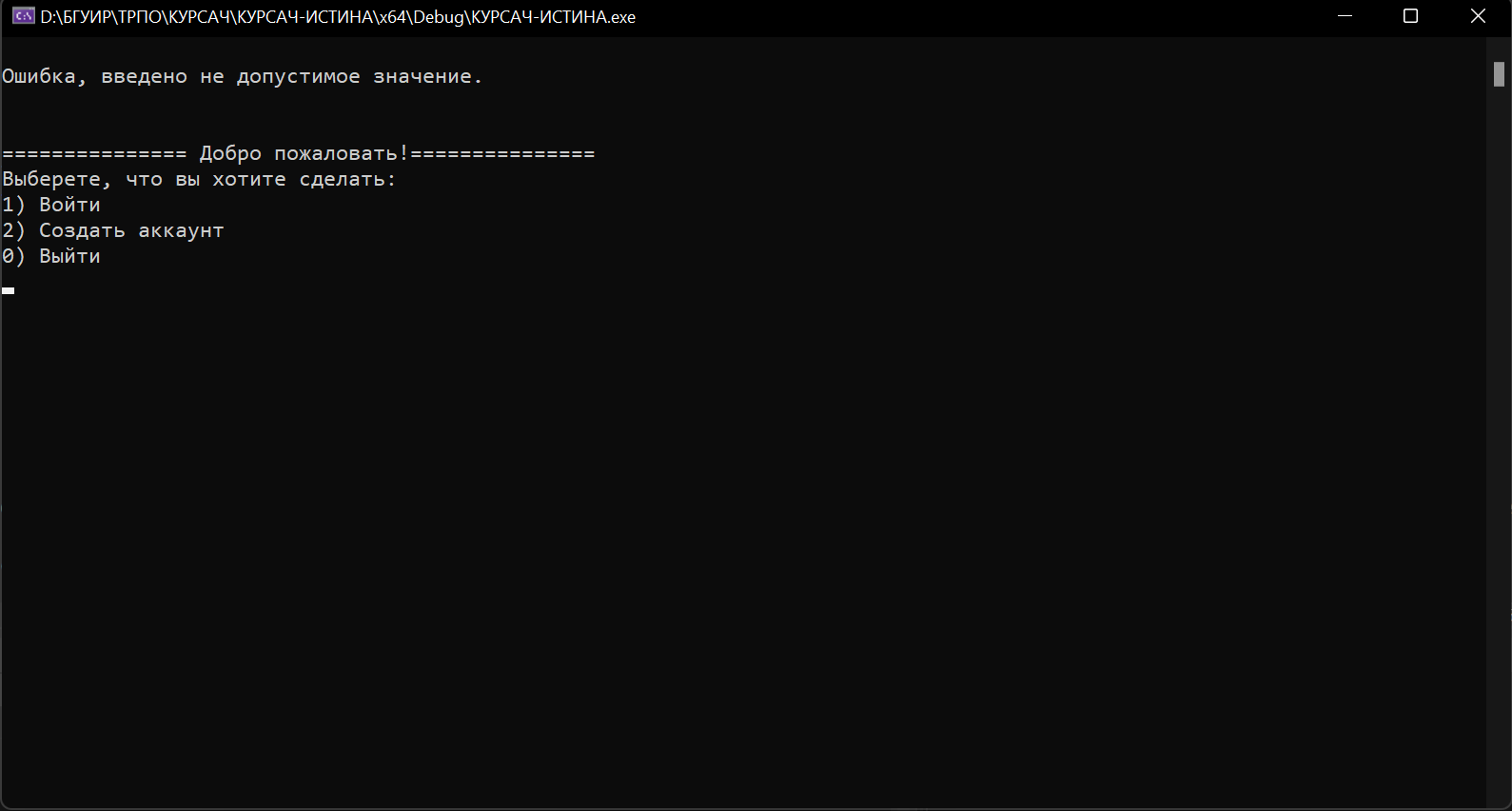


Рисунок 3.4 – Результат выбора пункта, которого нету

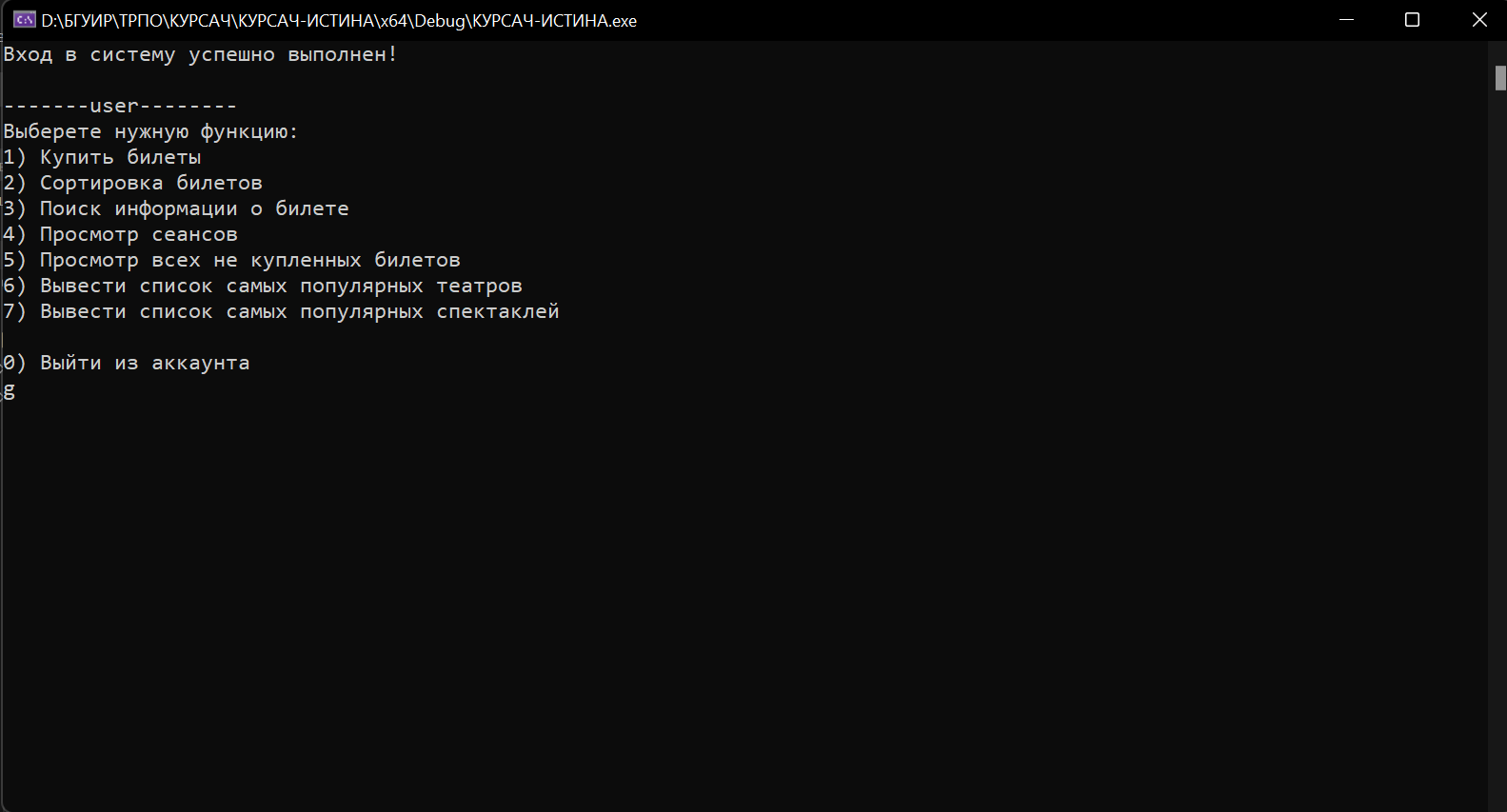


Рисунок 3.5 – Ввод нечислового значения в меню пользователя

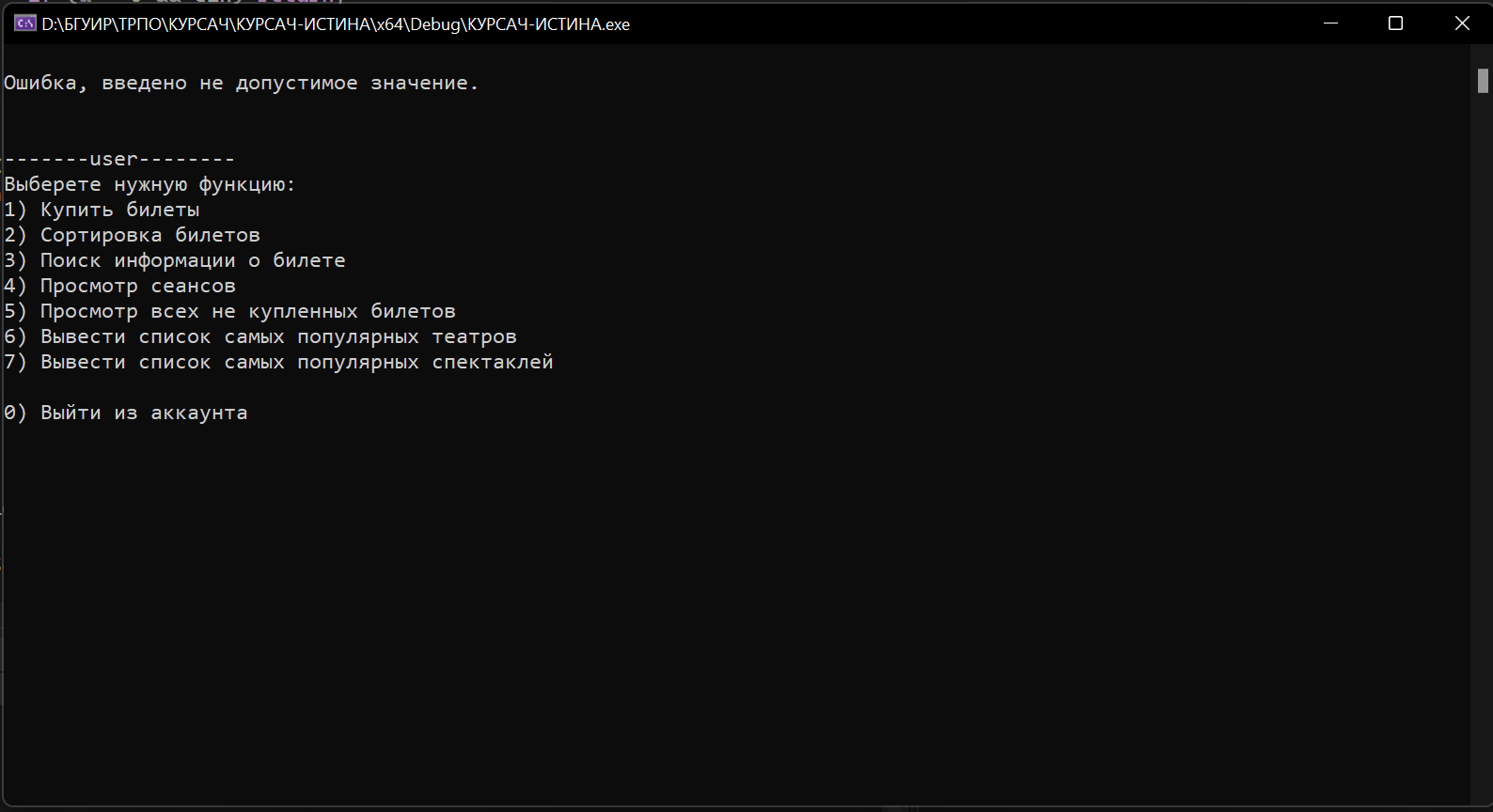


Рисунок 3.6 – Результат ввода нечислового значения в меню пользователя

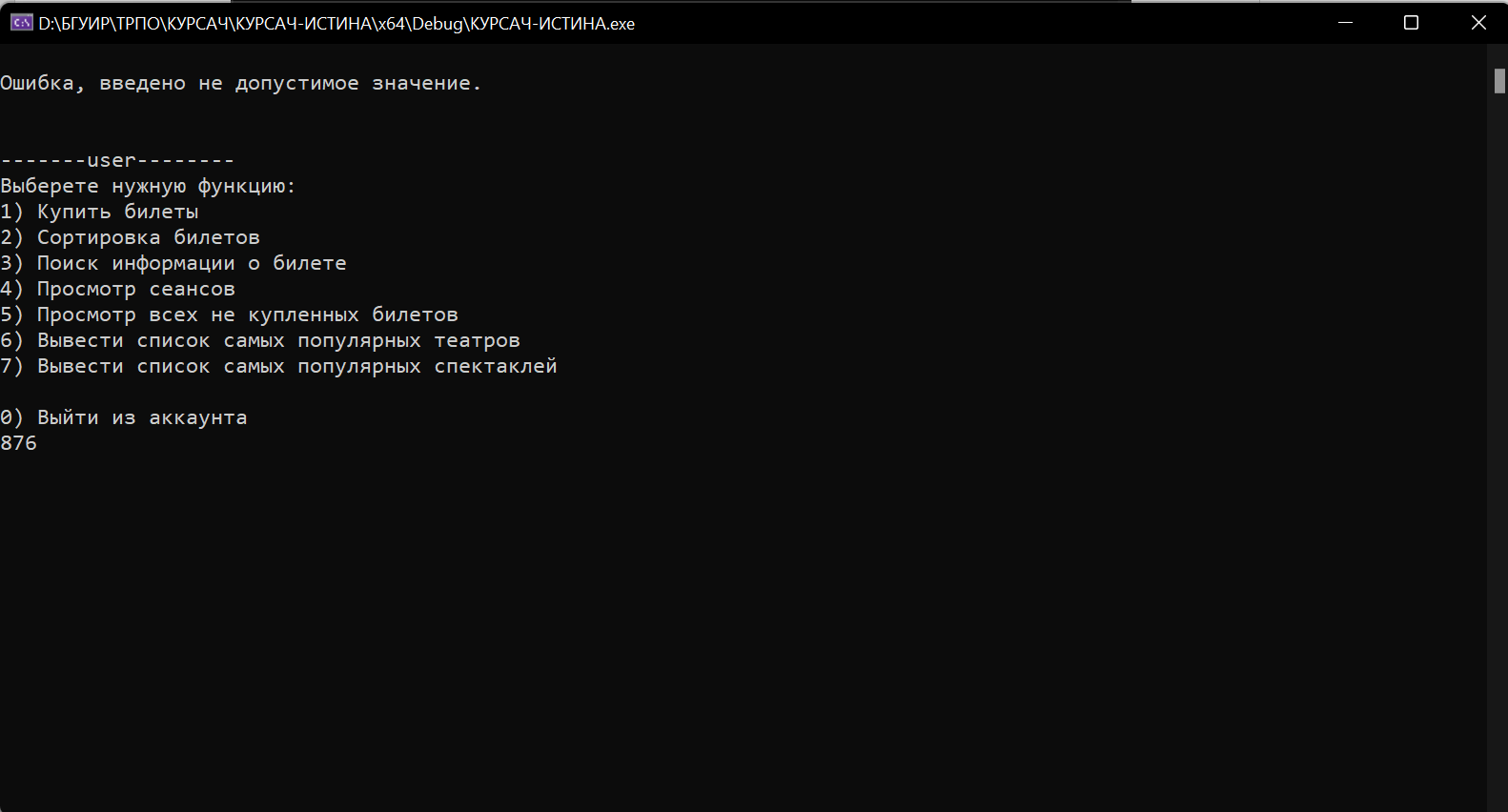


Рисунок 3.7 – Выбора пункта, которого нету в меню пользователя

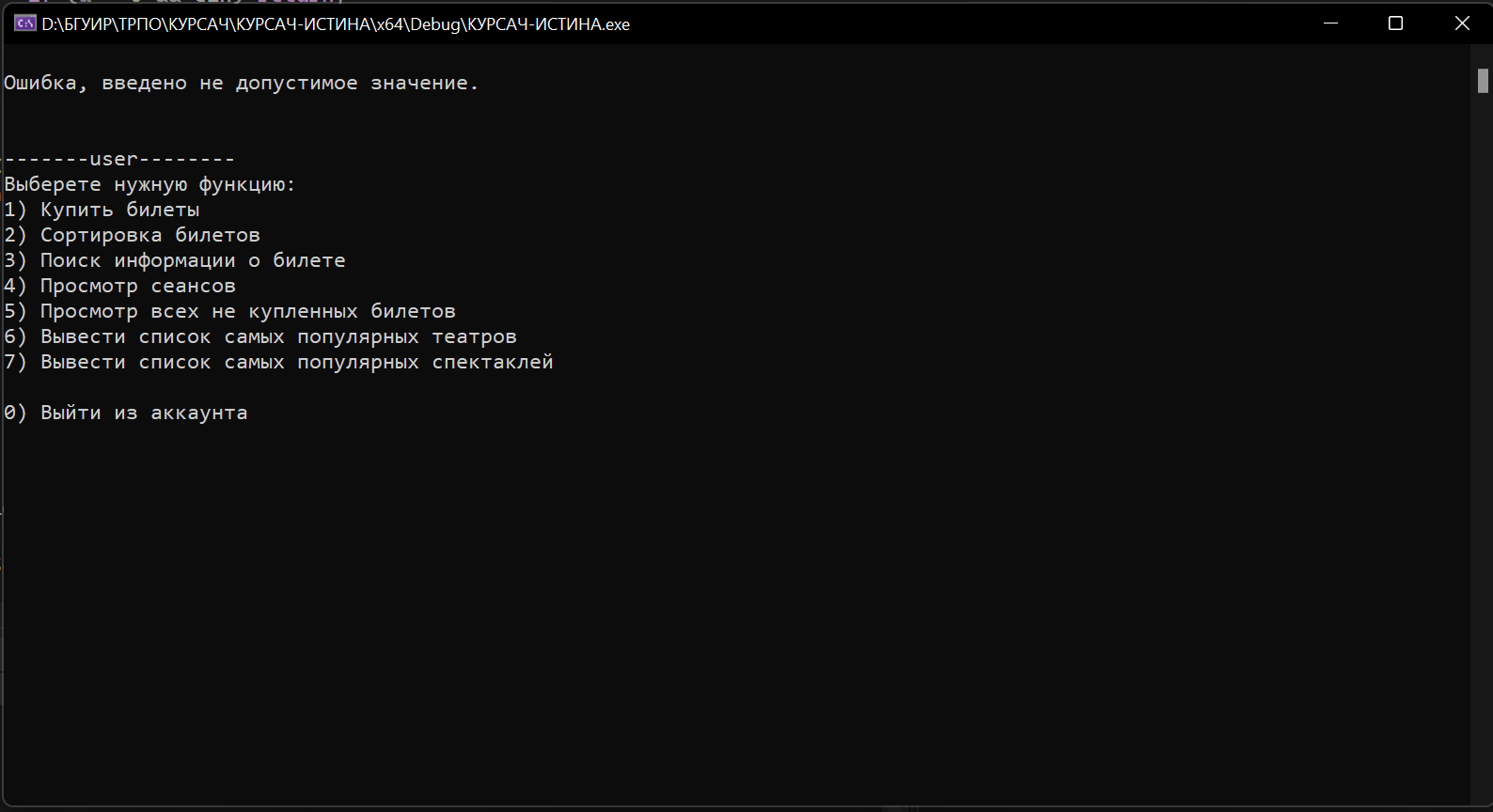


Рисунок 3.8 – Результат выбора пункта, которого нету в меню пользователя

# **Инструкция по развертыванию приложения и сквозной тестовый пример**

### Авторизация

При запуске программы нас встречает начальное окно (см. рисунок 4.1). Если у вас уже имеется аккаунт, нужно выбрать пункт “Войти”, если у вас нету аккаунта выберите пункт “ Создать аккаунт если вам требуется выключить программу, выберите пункт “ Выйти”.

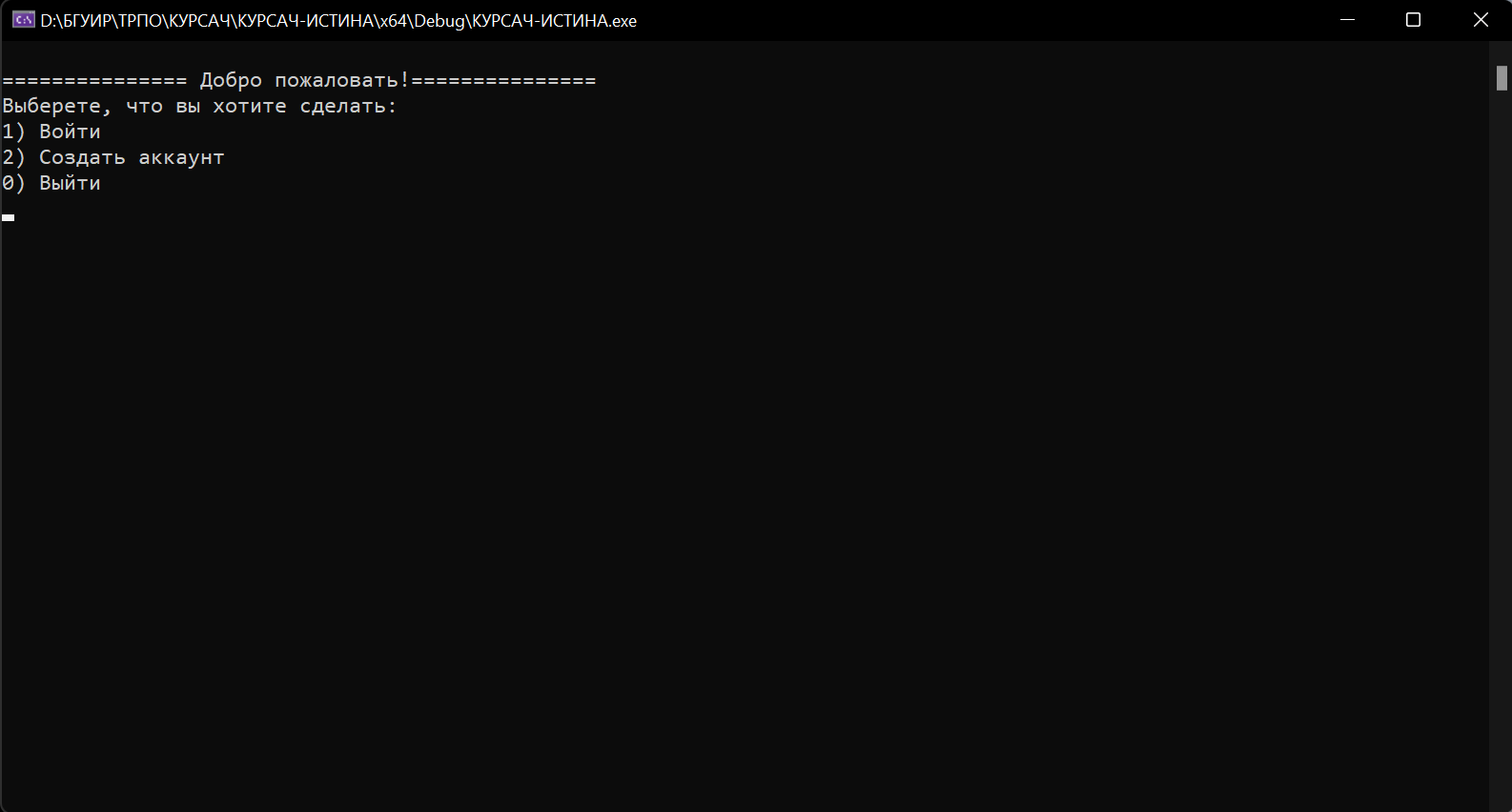


Рисунок 4.1 - Начальное окно

При выборе пункта “Войти”, вас попросят ввести ваш логин и пароль (см. рисунок 4.2). Введите ваш логин и пароль в соответствующих полях. Если вход прошел успешно, вы увидите соответствующее сообщение и перед вами появятся функции в зависимости от вашего статуса, для пользователя (см. рисунок 4.3) и администратора (см. рисунок 4.4), если же нет, требуется повторить ввод (см. рисунок 4.5).



Рисунок 4.2 – Ввод логина и пароля

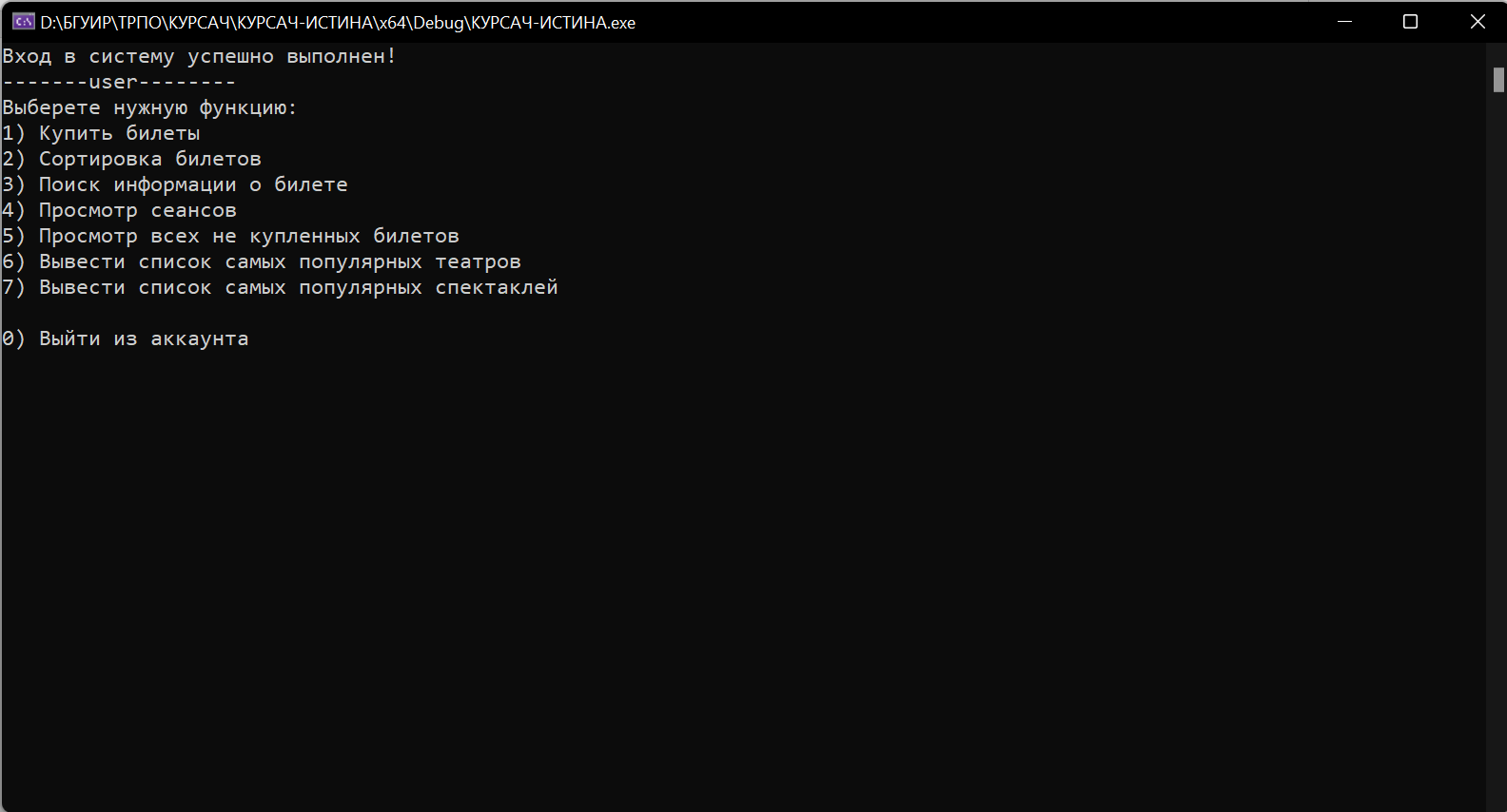


Рисунок 4.3 – Успешный вход пользователя

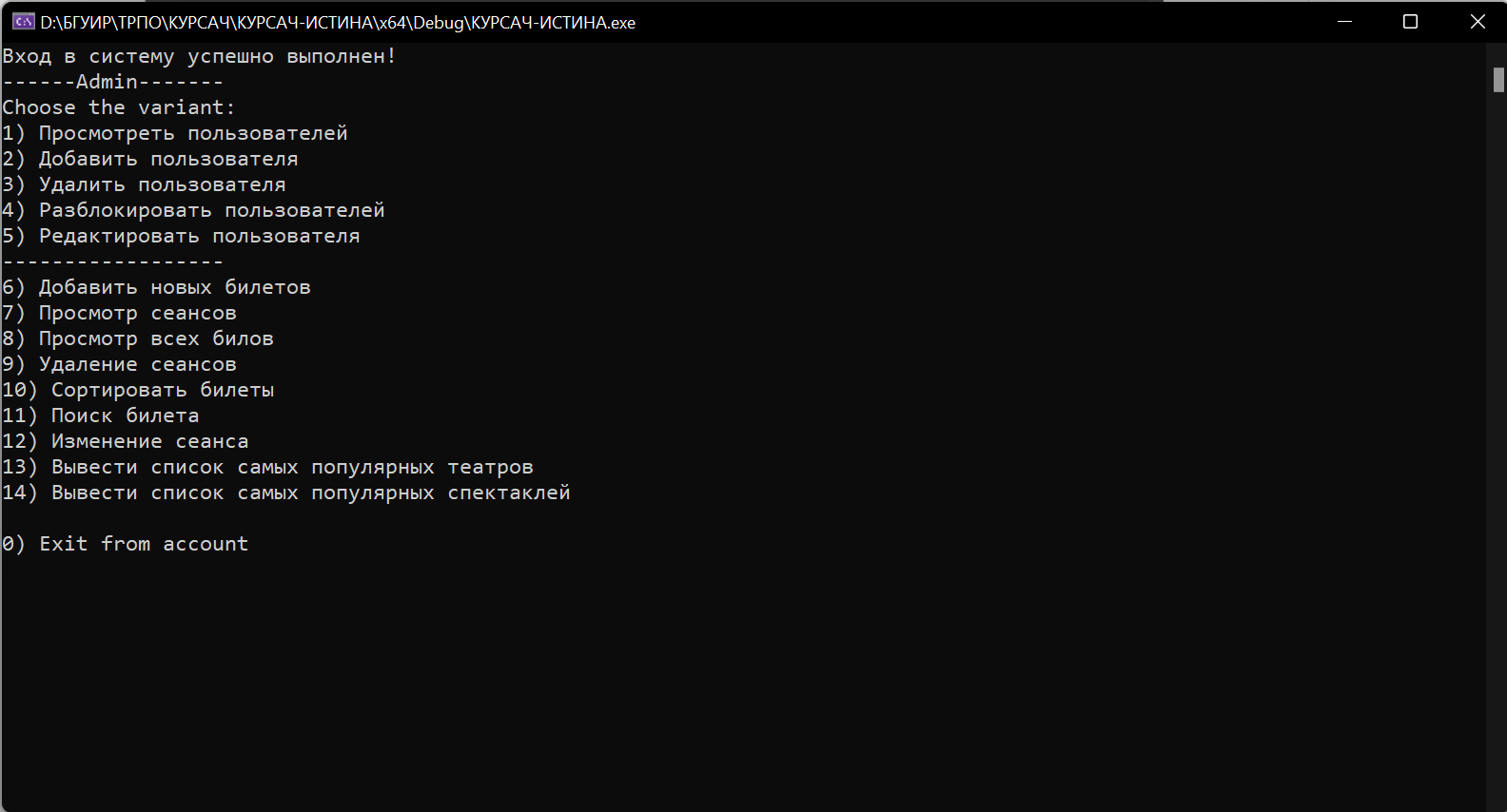


Рисунок 4.4 – Успешный вход администратора

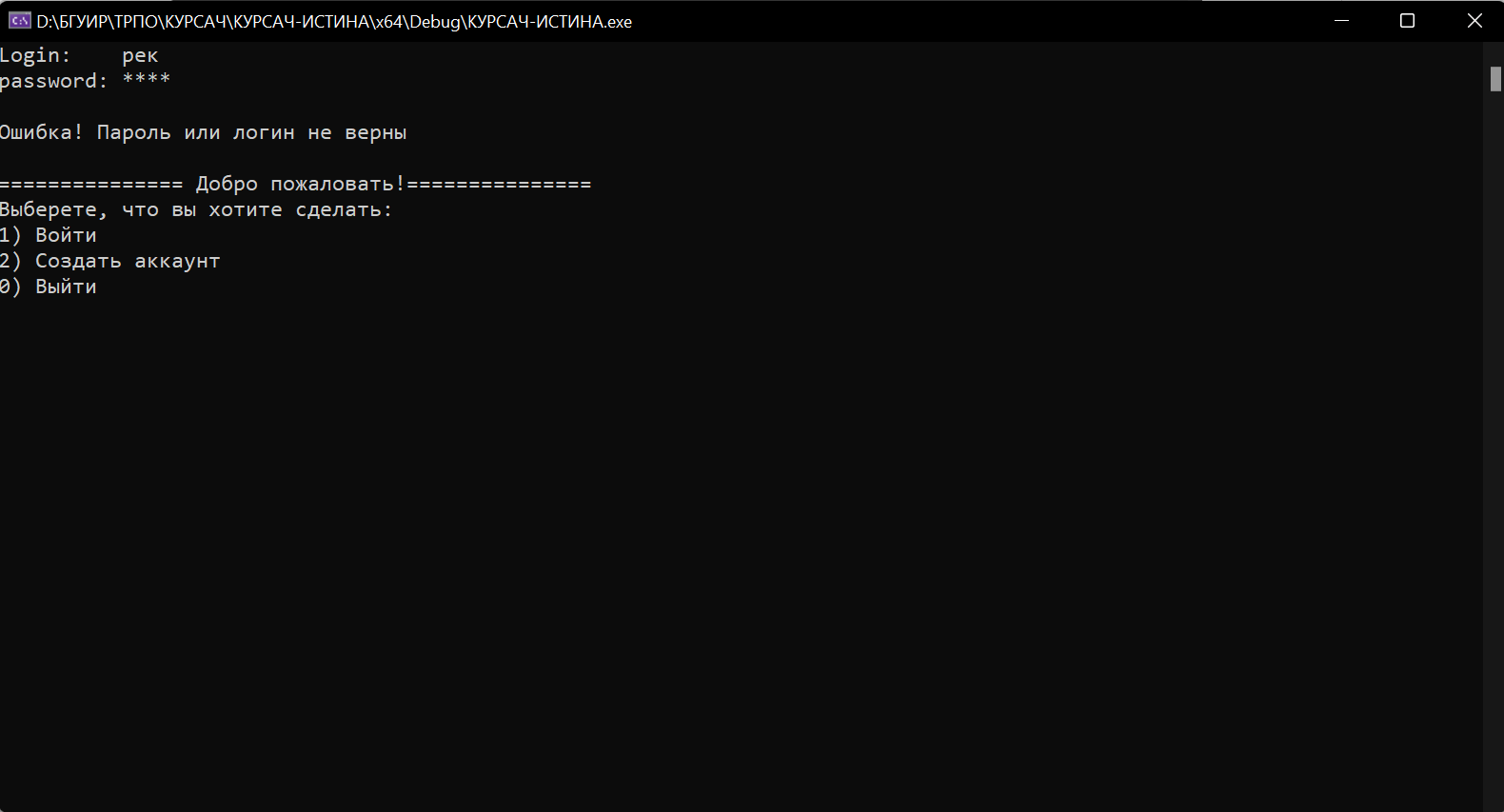


Рисунок 4.5 – Ошибка входа

### Модуль администратора

После успешной авторизации, если вы зашли в аккаунт админа, перед вами откроется окно различных функций администрации (см. рисунок 4.6). Функции условно можно разделить на 3 группы: функции взаимодействия с сеансами и билетами, функции взаимодействия с пользователями и функция выхода. Для взаимодействия с функцией введите соответствующий ей номер в консоль.

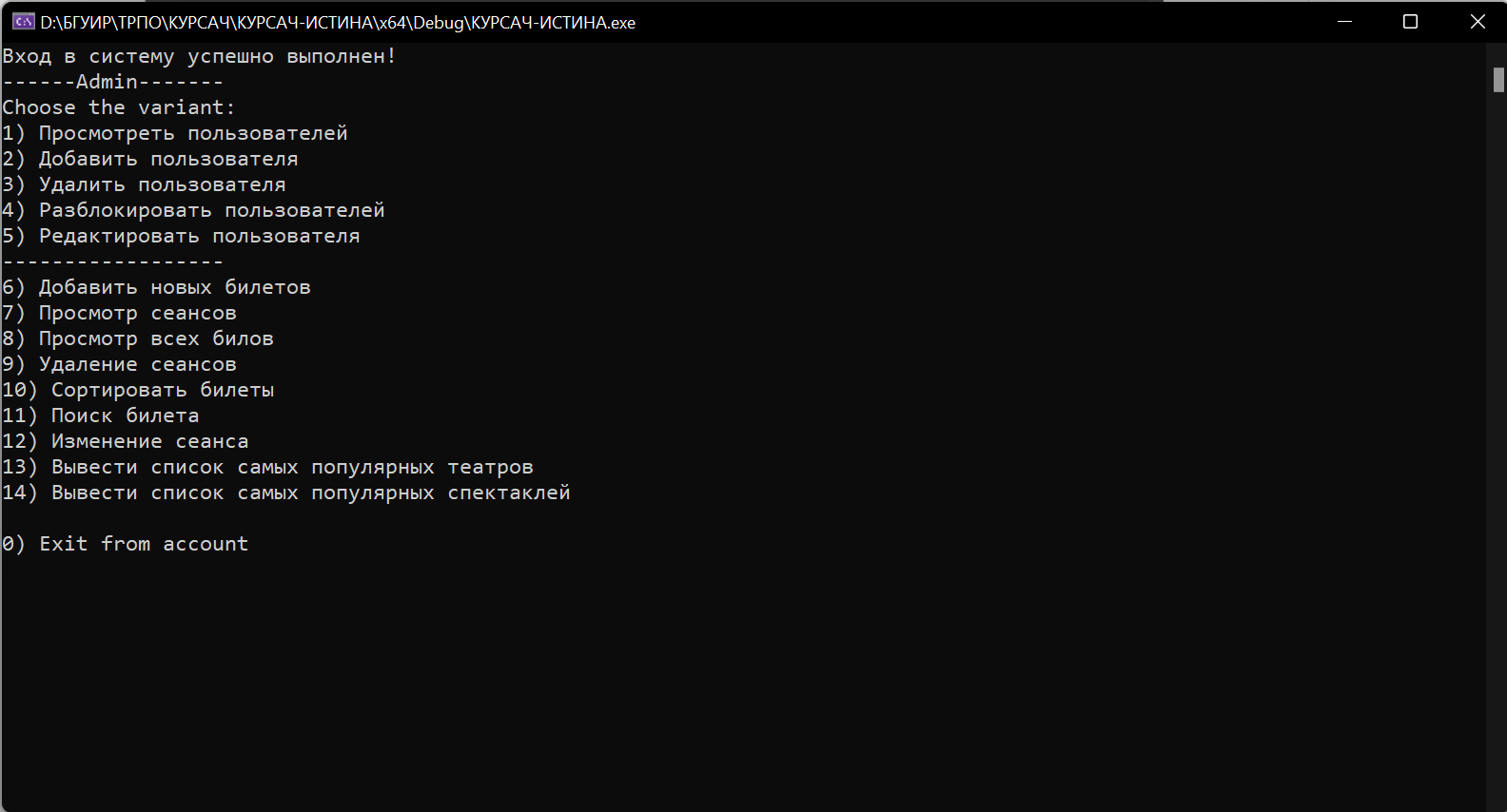


Рисунок 4.6 – Интерфейс администратора

### Модуль пользователя

После успешной авторизации, если вы зашли в аккаунт пользователя, перед вами откроется окно различных функций базового пользователя (см. рисунок 4.7). Функции пользователя сильно ограничены и направлены на минимальные изменения данных. Для взаимодействия с функцией введите соответствующий ей номер в консоль.

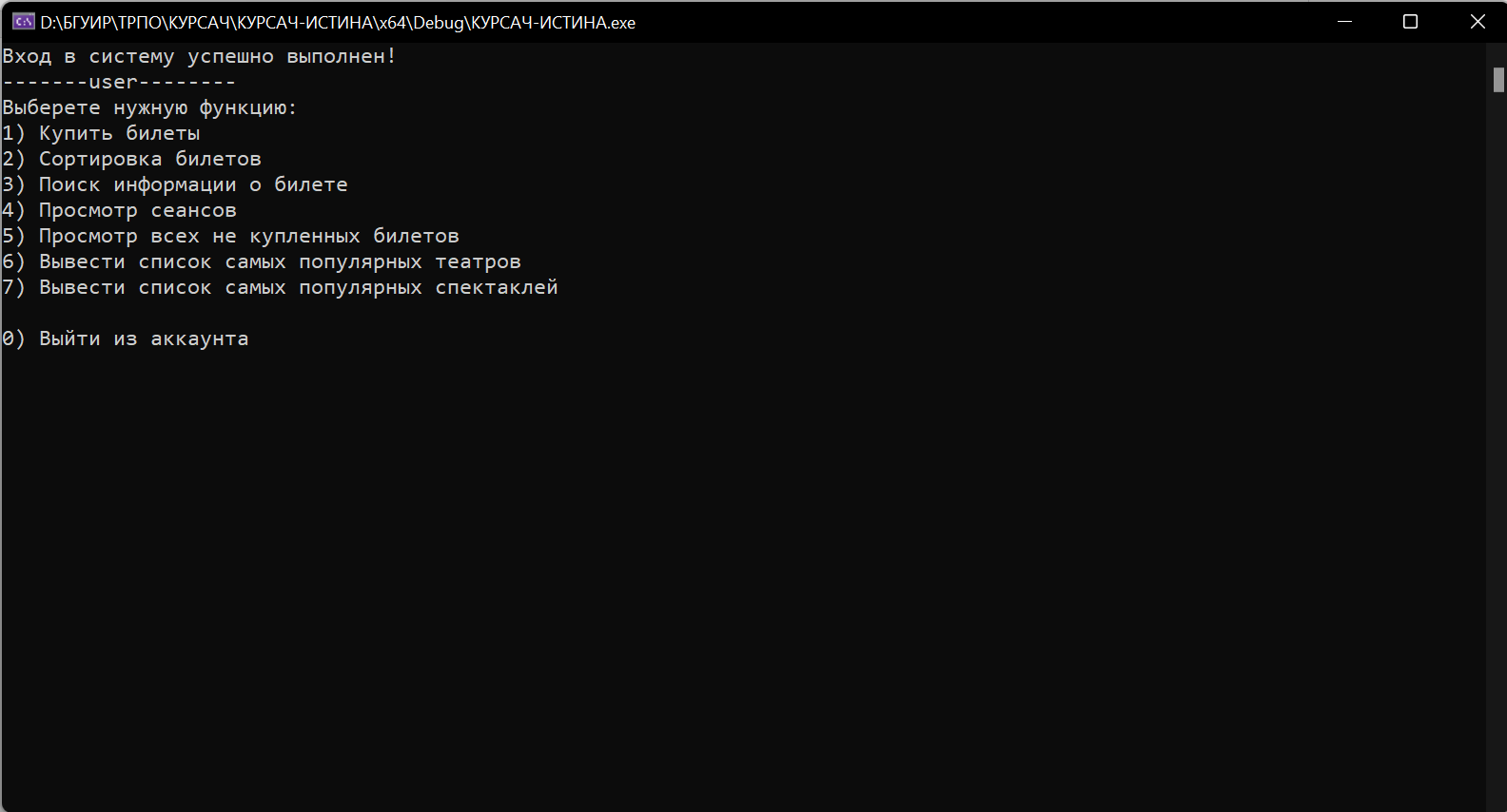


Рисунок 4.7 – Интерфейс пользователя

# **Заключение**

По итогу разработки программного средства по учету билетов в кассах театров, было реализовано рабочее приложение, содержащее в себе функции по взаимодействию с сеансами и билетами: добавление, просмотр, удаление, сортировка, изменение информации и т.д.; функции по взаимодействию с пользователями: создание аккаунтов, удаление аккаунтов, просмотр списков админов и пользователей.

Если говорить про программную реализацию, то была изучена работа с потоками данных и классами.

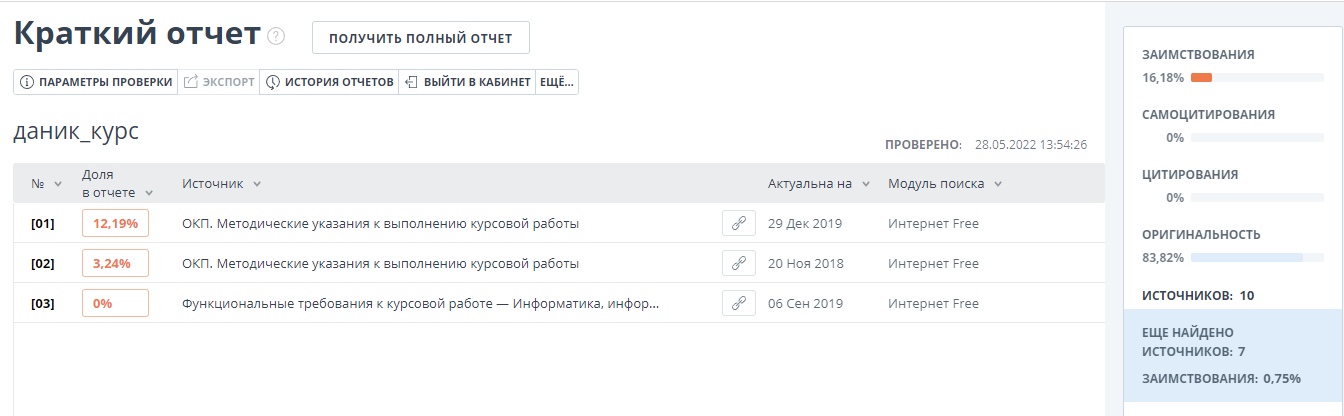
В пояснительной записке были затронуты вопросы надобности, актуальности и реализации приложения. Были расписаны назначения основных функций приложения. Предоставлены схемы работы приложения, алгоритмов основных функций.

Также была описана краткая инструкция по авторизации и взаимодействию клиентов и админов с приложением.

# **Список использованных источников**

1. Навроцкий, А. А. Основы алгоритмизации и программирования в среде Visual C++ : учеб.-метод. пособие / А. А. Навроцкий. – Минск : БГУИР, 2014. – 160 с.
2. Шилдт, Г. С++ Базовый курс / Г. Шилдт ; пер. с англ. – 3-е изд. – М. : Изд. дом «Вильямс», 2015. – 624 с.
3. Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл ; пер. с англ. – М. : Русская редакция, 2010. – 896 с.
4. Документация по Visual Studio [Электронный ресурс]. – Режим доступа : <https://docs.microsoft.com/ru-ru/visualstudio/ide/?view=vs-2017>.
5. Todd Hoff C++ Coding Standard [Электронный ресурс]. – Режим доступа : [http://www.possibility.com/Cpp/c++\_coding\_standards.pdf](http://www.possibility.com/Cpp/c%2B%2B_coding_standards.pdf).
6. Google C++ Style Guide [Электронный ресурс]. – Режим доступа : <https://google.github.io/styleguide/cppguide.html>.
7. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем.
8. Доманов, А.Т. Стандарт предприятия СТП 01-2017 / А. Т. Доманов, Н. И. Сорока. – Минск : БГУИР, 2017. – 169 с.

# **ПРИЛОЖЕНИЕ А Результат “Антиплагиат”**



# **ПРИЛОЖЕНИЕ Б Листинг основных функций приложения**

bool is\_true = false;

void addNewBilets(GeneralInformationBilets\* all\_general\_bilets, int& koll\_general\_bilets, Bilet\* all\_bilets, int& koll\_bilets, int RESERVE\_SIZE) {

system("cls");

int row, place;

if (koll\_general\_bilets >= RESERVE\_SIZE) {

cout << "Ошибка! Достигнут лимит количества билетов\n"; return;

}

cout << "Введите год билета: ";

cin >>all\_general\_bilets[koll\_general\_bilets].year;

CikleWhileForaddNewTicket2(all\_general\_bilets[koll\_general\_bilets].month, 0, 13, "Введите месяц билета (от 1 до 12): ");

CikleWhileForaddNewTicket2(all\_general\_bilets[koll\_general\_bilets].day, 0, 31, "Введите день билета (от 1 до 30): ");

CikleWhileForaddNewTicket(row, "\nВведите сколько рядов будет в зале: (>0): ");

CikleWhileForaddNewTicket(place, "\nВведите сколько мест будет в ряде: (>0): ");

CikleWhileForaddNewTicket(all\_general\_bilets[koll\_general\_bilets].price, "Введите цену одного билета: (>0): ");

all\_general\_bilets[koll\_general\_bilets].kollOst = (row \* place);

cout << "Введите название театра:\n";

cin.ignore((numeric\_limits<streamsize>::max)(), '\n');

getline(cin, all\_general\_bilets[koll\_general\_bilets].teatr);

cout << "Введите название спектакля:\n";

getline(cin, all\_general\_bilets[koll\_general\_bilets].spectacl);

system("cls");

all\_general\_bilets[koll\_general\_bilets].kollBuy = 0;

for (int i = 0; i < koll\_general\_bilets; i++)

if (all\_general\_bilets[koll\_general\_bilets] == all\_general\_bilets[i]) {

cout << "Ошибка. Такой сеанс уже существует\n";

return;

}

if (koll\_bilets == 0) { all\_bilets[koll\_bilets - 1].nomer = 0; }

for (int n = 0; n < row; n++)

for (int y = 0; y < place; y++) {

all\_bilets[koll\_bilets].addBilet(all\_general\_bilets[koll\_general\_bilets].teatr, all\_general\_bilets[koll\_general\_bilets].spectacl, (all\_bilets[koll\_bilets-1].nomer+1), all\_general\_bilets[koll\_general\_bilets].day, all\_general\_bilets[koll\_general\_bilets].month, all\_general\_bilets[koll\_general\_bilets].year, all\_general\_bilets[koll\_general\_bilets].price, n+1, y+1);

koll\_bilets++;

}

cout << "Билеты успешно добавлены!\n";

koll\_general\_bilets++;

}

void Buy(Bilet\* all\_bilets, int koll\_bilets, int amount, GeneralInformationBilets\* all\_general\_bilets, int choice) {

int nomer, prowerka;

cout << "Введите номе билета для покупки\n";

for (int i = 0; i < amount; i++) {

prowerka = 0;

cout << "Ввдите номер " << i + 1 << " билета, который вы хотите купить:";

cin >>nomer;

for (int b = 0; b < koll\_bilets; b++) {

if (all\_bilets[b].nomer == nomer && all\_bilets[b].buy == 0 && all\_bilets[b].day == all\_general\_bilets[choice].day && all\_bilets[b].month == all\_general\_bilets[choice].month && all\_bilets[b].year == all\_general\_bilets[choice].year && all\_bilets[b].teatr == all\_general\_bilets[choice].teatr && all\_bilets[b].spectacl == all\_general\_bilets[choice].spectacl) {

all\_bilets[b].buy = 1;

cout << "Вы купили бтлет:\n----------------------\n";

all\_bilets[b].ptintBilet();

cout << "----------------------\n";

prowerka++;

break;

}

}

if (prowerka == 0) {

cout << "Ошибка. Номер билета введен не верно или он уже куплен или он принадлежит к другой серии. Повторите попытку";

i--;

}

}

}

void buyBilets(GeneralInformationBilets\* all\_general\_bilets, int koll\_general\_bilets, Bilet\* all\_bilets, int koll\_bilets) {

system("cls");

int choice, amount;

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

while (true) {

CikleWhileForaddNewTicket2(choice, -3, koll\_general\_bilets + 1, "Введите номер серии, из котороы вы хотите купить билет\nДля настройки фильтров билетов введите -1;\nВернуться на главную: -2\n");

switch (choice)

{

default:

cout << "Введите количество покупаемых билетов: ";

cin >>amount;

if (amount < 0 || amount > all\_general\_bilets[choice - 1].kollOst) {

system("cls");

cout << "Ошибка! Неверно введено количество билетов\n";

break;

}

choice--;

all\_general\_bilets[choice].kollOst -= amount;

all\_general\_bilets[choice].kollBuy += amount;

system("cls");

viewBiletsForBuy(all\_bilets, koll\_bilets, all\_general\_bilets, choice);

Buy(all\_bilets, koll\_bilets, amount, all\_general\_bilets, choice);

system("cls");

cout << "Билеты успешно куплены!\n";

return;

case -1:

system("cls");

SortBilets(all\_general\_bilets, koll\_general\_bilets);

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

break;

case -2:

system("cls");

return;

}

}

}

void ChangeBilets(GeneralInformationBilets\* all\_general\_bilets, int koll\_general\_bilets, Bilet\* all\_bilets, int koll\_bilets) {

system("cls");

int schot, wish, month, day, year, price;

string teatr, spectacl;

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

while (true) {

CikleWhileForaddNewTicket2(wish, -2, koll\_general\_bilets + 1, "Выберете какай сеанс вы хотите озменить\nЕсли желаите отсортировать билеты нажмите -1, если выйти 0:\n");

if (wish == 0) {

system("cls");

return;

}

else if (wish == -1) {

system("cls");

SortBilets(all\_general\_bilets, koll\_general\_bilets);

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

}

else break;

}

system("cls");

wish--;

year = all\_general\_bilets[wish].year;

month = all\_general\_bilets[wish].month;

day = all\_general\_bilets[wish].day;

price = all\_general\_bilets[wish].price;

teatr = all\_general\_bilets[wish].teatr;

spectacl = all\_general\_bilets[wish].spectacl;

while (true) {

all\_general\_bilets[wish].ptintBilet();

switch (menuForChangeBilets()) {

case 1: cout << "Введите год: "; cin >>year; break;

case 2: CikleWhileForaddNewTicket2(month, 0, 13, "Введите новый месяц сеанса (от 1 до 12): "); break;

case 3: CikleWhileForaddNewTicket2(day, 0, 31, "Введите новый день сеанса (от 1 до 30): "); break;

case 4: CikleWhileForaddNewTicket(price, "Введите новую цену одного билета: (>0): "); break;

case 5: cin.ignore((numeric\_limits<streamsize>::max)(), '\n'); cout << "Введите название театра :"; getline(cin, teatr); break;

case 6: cin.ignore((numeric\_limits<streamsize>::max)(), '\n'); cout << "Введите название театра :"; getline(cin, spectacl); break;

case 0: return;

}

schot = 0;

for (int h = 0; h < koll\_general\_bilets; h++)

if (h != wish && all\_general\_bilets[h].spectacl == all\_general\_bilets[wish].spectacl && all\_general\_bilets[h].teatr == all\_general\_bilets[wish].teatr && all\_general\_bilets[h].year == all\_general\_bilets[wish].year && all\_general\_bilets[h].month == all\_general\_bilets[wish].month && all\_general\_bilets[h].day == all\_general\_bilets[wish].day && all\_general\_bilets[h].price == all\_general\_bilets[wish].price) {

cout << "\nОшибка! Такой сеанс уже существуетю Последние изменения не будут учитываться\n";

schot++;

break;

}

if (schot == 0) {

for (int h = 0; h < koll\_bilets; h++)

if (all\_bilets[h].spectacl == all\_general\_bilets[wish].spectacl && all\_bilets[h].teatr == all\_general\_bilets[wish].teatr && all\_bilets[h].year == all\_general\_bilets[wish].year && all\_bilets[h].month == all\_general\_bilets[wish].month && all\_bilets[h].day == all\_general\_bilets[wish].day && all\_bilets[h].price == all\_general\_bilets[wish].price) {

all\_bilets[h].year = year;

all\_bilets[h].month = month;

all\_bilets[h].day = day;

all\_bilets[h].price = price;

all\_bilets[h].teatr = teatr;

all\_bilets[h].spectacl = spectacl;

}

all\_general\_bilets[wish].year = year;

all\_general\_bilets[wish].month = month;

all\_general\_bilets[wish].day = day;

all\_general\_bilets[wish].price = price;

all\_general\_bilets[wish].teatr = teatr;

all\_general\_bilets[wish].spectacl = spectacl;

system("cls");

cout << "Данные успешно изменены\n\n";

}

}

}

void deleteBilet(GeneralInformationBilets\* all\_general\_bilets, int i, Bilet\* all\_bilets, int& koll\_bilets) {

int z = 0;

for (; z < (koll\_bilets - 1); z++)

if (all\_bilets[z].day == all\_general\_bilets[i].day

&& all\_bilets[z].month == all\_general\_bilets[i].month

&& all\_bilets[z].year == all\_general\_bilets[i].year

&& all\_bilets[z].teatr == all\_general\_bilets[i].teatr

&& all\_bilets[z].spectacl == all\_general\_bilets[i].spectacl) {

for (int r = z; r < (koll\_bilets-1); r++)

all\_bilets[r] = all\_bilets[r + 1];

koll\_bilets--;

z--;

}

z++;

if (all\_bilets[z].day == all\_general\_bilets[i].day && all\_bilets[z].month == all\_general\_bilets[i].month && all\_bilets[z].year == all\_general\_bilets[i].year && all\_bilets[z].teatr == all\_general\_bilets[i].teatr && all\_bilets[z].spectacl == all\_general\_bilets[i].spectacl) {

koll\_bilets--;

}

}

void deleteGeneralInformationBilets(GeneralInformationBilets\* all\_general\_bilets, int& koll\_general\_bilets, Bilet\* all\_bilets, int& koll\_bilets) {

system("cls");

int choice, delTicketNumber;

if (koll\_general\_bilets == 0) {

system("cls");

cout << "В базе данных нет ни одного билета!";

return;

}

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

while (true) {

CikleWhileForaddNewTicket2(choice, -2, koll\_general\_bilets + 1, "Введите номер сеанса, который хотите удалить\nДля настройки фильтров введите -1\nВернуться на гавную: 0\n");

switch (choice)

{

default:

system("cls");

deleteBilet(all\_general\_bilets, choice - 1, all\_bilets, koll\_bilets);

deleteGeneralBilet(all\_general\_bilets, choice - 1, koll\_general\_bilets);

cout << "Сеанс и билеты были успешно удален!\n";

return;

case -1:

system("cls");

SortBilets(all\_general\_bilets, koll\_general\_bilets);

viewGeneralInformationBilets(all\_general\_bilets, koll\_general\_bilets);

break;

case 0:

system("cls");

return;

}

}

}