

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра информационных технологий автоматизированных систем
Дисциплина основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
на тему

ИНФОРМАЦИОННАЯ СИСТЕМА ВОКЗАЛА

БГУИР КР 1-53 01 02 219

Студент
Руководитель

С. В. Марковцев
А. К. Ючков

Минск 2024

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись)

2024г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Марковцеву Станиславу Вадимовичу

1. Тема проекта Информационная система вокзала
2. Срок сдачи студентом законченного проекта 23 декабря 2024 г.
3. Исходные данные к проекту _____
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):
 - Введение
 - 1. Анализ технического задания на курсовое проектирование
 - 2. Проектирование информационной системы вокзала
 - 3. Реализация информационной системы
 - Заключение
 - Список использованных источников
 - Приложения
 - Ведомость курсовой работы
5. Консультант по проекту (с обозначением разделов проекта) Ючков А. К.
6. Дата выдачи задания 15 сентября 2024 г.
7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
 - раздел 1 к 16.10 – 30%
 - раздел 2 к 15.11 – 60%
 - раздел 3 к 15.12 – 90%
 - оформление пояснительной записки и графических
 - материалов к 23.12 – 100%
 - Защита курсового проекта с **23.12 по 26.12.2024**

Руководитель

(подпись)

Ючков А. К.

Задание принял к исполнению

(подпись)

Марковцев С. В.

СОДЕРЖАНИЕ

Введение	4
1 Анализ технического задания на курсовое проектирование	5
1.1 Анализ предметной области	5
1.2 Обзор систем-аналогов	6
1.3 Постановка задачи на курсовое проектирование	9
2 Проектирование информационной системы вокзала	11
2.1 Разработка функционала системы	11
2.2 Разработка иерархии классов	12
2.3 Хранение данных	13
3 Реализация информационной системы	16
3.1 Описание работы системы	16
Заключение	29
Список использованных источников	30
Приложение А (обязательное) Фрагмент исходного кода приложения .	31
Приложение Б (обязательное) Блок-схема алгоритма входа в приложение	44
Ведомость курсовой работы	46

ВВЕДЕНИЕ

С течением времени все больше систем начали становиться более комплексными и сложными в реализации. Изначально процесс создания любой информационной системы предполагал собой последовательную поэтапную разработку, которая имеет однонаправленный характер. Из-за этого возникла необходимость в рассмотрении новых подходов для более эффективного проектирования и разработки системы. В настоящее время наиболее перспективным можно выделить объектно-ориентированный подход проектирования.

Отличительной чертой модели объектно-ориентированного проектирования является отсутствие строгой последовательности в выполнении стадий как в прямом, так и в обратном направлениях процесса проектирования по отдельным компонентам.

Основное преимущество объектно-ориентированного подхода состоит в упрощении проектирования информационной системы при наличии типовых проектных решений по отдельным компонентам. Также наличие инкапсуляции, наследования и полиморфизма у объекта позволяет его легко модифицировать, поскольку модификация касается лишь отдельных компонент из-за их параллельности и автономности.

Объектно-ориентированные системы можно рассматривать как совокупность независимых объектов. При изменении реализации какого-нибудь объекта или же при добавлении этому объекту неких иных функций другие объекты системы не будут подвержены случайным изменениям. Темой курсового проекта является информационная система вокзала, которая позволит облегчить получение информации о поездах и поездках людей.

Целью курсового проекта является разработка иерархии классов с включенными в них полями и методами для обработки атрибутов, которая будет вспомогательным элементом для взаимодействия пользователя с данными вокзала. Также целью проекта является развитие навыков самостоятельной и творческой работы и закрепление навыков работы на языке C++.

Основными задачами данной работы являются – разработка структуры иерархии классов по заданной тематике, разработка схемы алгоритмов реализации отдельных компонентов, проведение тестирования приложения.

1 АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

1.1 Анализ предметной области

На сегодняшний день не существует аспектов работы вокзалов, которые бы не были связаны с применением информационных технологий. Благодаря современным информационным системам деятельность вокзалов, как узловых точек транспортной инфраструктуры, осуществляется эффективно и с высокой степенью автоматизации. Эти системы позволяют оптимизировать различные процессы, улучшить качество обслуживания пассажиров и повысить безопасность на объектах.

Информационные технологии и компьютеризация вокзалов позволяют усовершенствовать и ускорить процессы управления пассажирскими потоками, обработки данных, организации транспортных расписаний и других важных аспектов. Полная или частичная автоматизация этих процессов позволяет существенно сократить ручной труд, улучшить взаимодействие между сотрудниками вокзала, перевозчиками и пассажирами, а также повысить оперативность работы.

Информационные системы вокзалов являются важным ресурсом для улучшения взаимодействия пассажиров с кассами, операторскими службами, диспетчерами, а также для информирования о расписании транспорта и других важных сведениях. Несмотря на это, по экспертным оценкам, общий уровень информатизации вокзалов в современных условиях также нуждается в дальнейшем развитии и совершенствовании.

Основные аспекты информационной системы вокзала включают в себя следующие функции:

- управление билетами: система должна обеспечивать возможность покупки и бронирования билетов на различные виды транспорта, включая поезда, автобусы и самолёты. Важны функции расчета стоимости, выбора мест, а также возможность возврата и обмена билетов. Система должна интегрироваться с электронными платежами и обеспечивать поддержку различных способов оплаты.

- информация о расписании: информационная система вокзала должна предоставлять актуальные данные о расписании отправок и прибытий транспорта, обновлять эту информацию в режиме реального времени и информировать пассажиров о любых изменениях. Это включает работу с системами табло, мобильными приложениями и веб-сайтами вокзалов.

– управление пассажирскими потоками: система должна помогать в координации потоков пассажиров, обеспечивать эффективное распределение пассажиров по зонам ожидания, платформам и терминалам. Информационные технологии могут помочь прогнозировать загрузку и предотвращать перегрузки, оптимизируя работу вокзала в пиковые часы.

– управление инфраструктурой вокзала: система должна содержать данные о состоянии и готовности инфраструктуры вокзала, включая платформы, залы ожидания, эскалаторы, лифты и другие объекты. Важно иметь возможность планирования и контроля за техническим обслуживанием оборудования, чтобы избежать непредвиденных сбоев и обеспечить бесперебойную работу.

– управление клиентскими данными: система должна хранить информацию о пассажирах, включая историю покупок билетов, предпочтения и другие важные данные. Это позволит вокзалу предоставлять персонализированные услуги, повышать лояльность клиентов и улучшать качество обслуживания.

– безопасность и контроль доступа: информационные системы вокзалов должны включать средства контроля доступа, мониторинга территории и видеонаблюдения для обеспечения безопасности пассажиров и сотрудников. Это может включать интеграцию с системами распознавания лиц, сканирования багажа и контроля за соблюдением общественного порядка.

– финансовое управление: система должна обеспечивать учет и обработку финансовых операций, связанных с продажей билетов, арендой помещений, оплатой услуг и других финансовых аспектов работы вокзала. Необходимо наличие инструментов для анализа финансовых данных, составления отчетов и планирования бюджета.

– аналитика и отчетность: система должна предоставлять возможности для анализа и визуализации данных, связанных с работой вокзала. Это включает отчеты о пассажиропотоках, продажах билетов, эффективности работы инфраструктуры и других ключевых показателях, которые помогают улучшать работу вокзала и планировать развитие.

Эти аспекты информационной системы вокзала помогают обеспечить эффективное управление и повышают качество обслуживания пассажиров, создавая комфортные и безопасные условия для их передвижения.

1.2 Обзор систем-аналогов

В данном курсовом проекте необходимо реализовать информационную систему вокзала. Для создания информационной системы изучены веб-сайты с целью получения информации о необходимом функционале. Самым попу-

лярным сайтом в Беларуси по покупке билетов является <https://pass.rw.by/>. Рассмотрим его.

Допустим необходимо купить некоторое количество билетов на рейс Минск-Гомель. Для этого на главной странице необходимо ввести нужные данные – пункт отправления, пункт назначения, а также день поездки. (Рисунок 1.1)

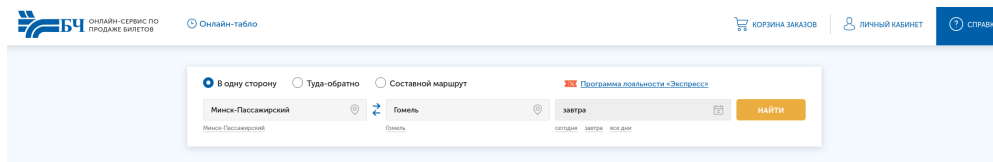
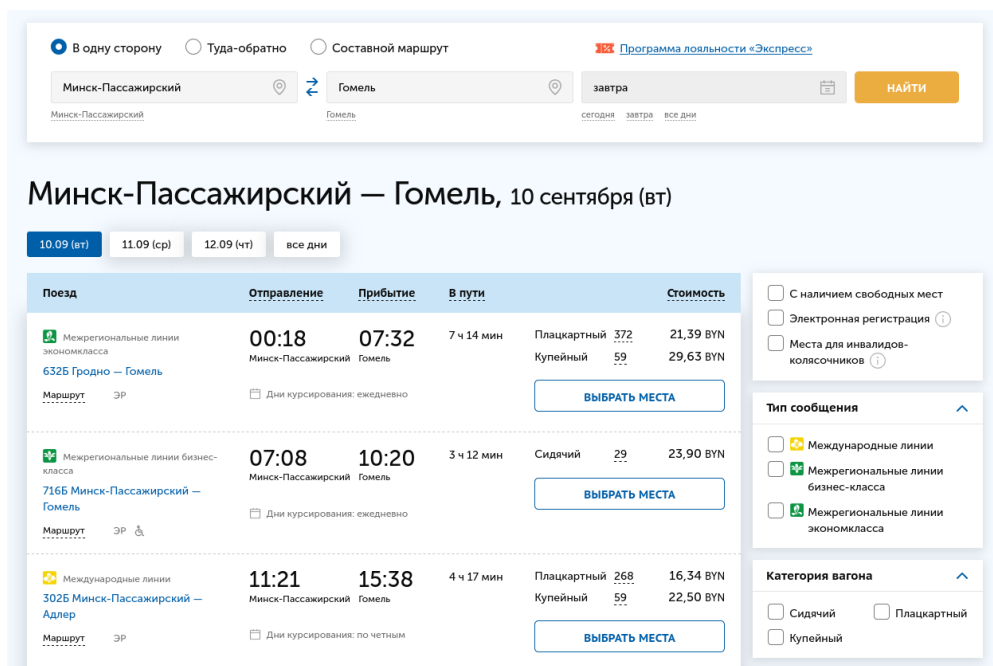


Рисунок 1.1 – Покупка билетов на сайте <https://pass.rw.by/>

Можем заметить что на данной странице осуществлен поиск по первым введенным буквам по городу. Введем данные которые нам подходят, а так же укажем дату(завтра). После чего необходимо нажать на кнопку «Найти». (Рисунок 1.2)



Поезд	Отправление	Прибытие	В пути	Стоимость
Межрегиональные линии экономкласса 632Б Гродно — Гомель	00:18	07:32	7 ч 14 мин	Плацкартный 372 21,39 BYN Купейный 59 29,63 BYN
Межрегиональные линии бизнес-класса 716Б Минск-Пассажирский — Гомель	07:08	10:20	3 ч 12 мин	Сидячий 29 23,90 BYN
Международные линии 302Б Минск-Пассажирский — Адлер	11:21	15:38	4 ч 17 мин	Плацкартный 268 16,34 BYN Купейный 59 22,50 BYN

Рисунок 1.2 – Страница выбранного направления

После ввода данных о поездке, сервис предлагает выбрать поезд. Так же указано количество мест и цена. Чтобы выбрать места необходимо нажать на кнопку «Выбрать места» для необходимого поезда. (Рисунок 1.3)

Выберите количество пассажиров

1

+

Полный

От 10 лет

0

+

Дети с местом

До 10 лет, с оплатой по льготному тарифу. Внимание! Проезд детей в возрасте до 10 лет без сопровождения взрослых не допускается

0

+

Дети без места

До 5 лет, бесплатные билеты без занятия отдельного места одновременно с оплатой билетов по «полному» тарифу

Выберите тип вагона

Сидячий

Мест: 36

23,90 / 47,81 BYN

Вагон №04 (2Р)

ЭР

Всего свободно мест – 1

47,81 BYN

Вагон №01 (2В)

ЭР

Всего свободно мест – 6

23,90 BYN

Вагон №02 (2В)

ЭР

Всего свободно мест – 10

23,90 BYN

Вагон №03 (2В)

ЭР

Всего свободно мест – 15

23,90 BYN

Вагон №04 (2В)

ЭР

Всего свободно мест – 6

23,90 BYN

Вагон №05 (2В)

ЭР

Всего свободно мест – 18

23,90 BYN

Рисунок 1.3 – Страница выбора вагона

После нажатия на кнопку «Выбрать места» необходимо выбрать вагон, в результате чего откроется окно выбора мест, где свободные места имеют белый фон, а занятые – серый. Также, можно не выбирать конкретное место. В таком случае оно будет предложено системой. (Рисунок 1.4)

Вагон №01 (2В)

ЭР

Всего свободно мест – 6

23,90 BYN

Выбор на схеме

Требования к местам

Чтобы выбрать места нажмите на свободное место (при заказе одного места) или первое и последнее свободные места необходимого диапазона. По желанию, задайте ярусность требуемых мест (верхнее/нижнее). Места будут зарезервированы после ввода данных пассажиров, проверки реквизитов поездки и активации команды «ОФОРМИТЬ ЗАКАЗ».

— недоступное место

— свободное место

— выбранное место

— выбранный диапазон мест

Перевозчик - БЧ; Принадлежность вагона - БЧ / БЧ

← Гомель

Рисунок 1.4 – Страница выбора места

После выбора места, необходимо ввести данные о пассажире. Также, предусмотрена система выбора двух и более пассажиров. Билеты в таком случае можно оформить либо на одного, либо на двух и более людей. (Рисунок 1.5)

После ввода данных о пассажире, заказ будет создан и помещен в корзину, которую необходимо оплатить в течении 20 минут. Эти 20 минут, билет будет зарезервирован и никто другой не сможет его взять.

8

Заполните данные

Внимание! При покупке проездных документов (билетов) для проезда по территории Республики Беларусь и в международном сообщении в страны СНГ, персональные данные пассажира следует вводить согласно документу, удостоверяющему личность, на основании которого будет совершаться поездка (на кириллице или латинице), для проезда в другие страны – на латинице.

Пассажир №1

Очистить форму

Тариф: Полный (Взрослый)

Фамилия *

Иванов

Имя *

Иван

Отчество

Иванович

Обязательно при наличии

Тип документа *

Паспорт гражданина Республик... ▼

Номер документа * ⓘ

MP1234567

☐ Подтверждаю, что с [правилами и особенностями](#) оформления заказа, его оплаты, оформления и переоформления проездного документа (билета), возврата неиспользованного проездного документа (билета), заказанного через Интернет, ознакомлен

ОФОРМИТЬ ЗАКАЗ

Рисунок 1.5 – Страница ввода данных о пассажире

1.3 Постановка задачи на курсовое проектирование

В данной курсовой работе необходимо разработать структуру иерархии классов информационной системы транспортной компании. Также необходимо отразить с помощью диаграмм структуру классов и их иерархии и с помощью блок-схемы отобразить ход работы алгоритма. Для реализации поставленных задач необходимо следующее:

- разработать иерархию классов, определить базовый и наследуемые классы;
- разработать и описать структуру каждого класса в отдельности, объявить поля и методы класса;
- реализовать программу для работы с иерархией классов.

Так как основным способом реализации консольного приложения в данной работе является объектно-ориентированное программирование на языке C++, то структура приложения предполагает наличие базового и производных классов, то есть их иерархию, а также наличие полей и методов, которые описывают класс и позволяют взаимодействовать между объектами, воздействовать на сам объект и выполнять объекту определённые функции.

По требованию пользователя должно быть выполнено следующее дей-

ствие:

- выбор маршрута рейса;
- выбор даты и времени рейса;
- изменение штата работников;
- демонстрация возможностей системы;
- возможность возвращения на предыдущий этап взаимодействия;
- безопасное завершение работы программы.

2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ ВОКЗАЛА

2.1 Разработка функционала системы

Для разработки данной информационной системы использовался уже ранее представленный язык программирования C++. В качестве *IDE (integrated development environment)* для работы с этим языком был выбран *Clion*. Эта программа предоставляет все необходимые возможности для комфортной разработки программного средства.

Данный язык был выбран так же с учетом того, что он предоставляет ряд готовых решений. Поэтому во время выполнения курсового проекта использовались следующие библиотеки:

- библиотека *iostream* – предоставляет инструменты для организации ввода и вывода данных в языке программирования C++. Основные компоненты включают *std::cin* для ввода, *std::cout* для вывода, а также *std::cerr* для отображения ошибок.

- библиотека *limits* – содержит шаблонный класс *std::numeric_limits*, предоставляющий информацию о свойствах числовых типов, таких как максимальное и минимальное значения, точность, поведение при переполнении.

- библиотека *sqlite3.h* – библиотека для работы с базой данных SQLite. Предоставляет функции и методы для создания, изменения, удаления таблиц и записи данных, а также выполнения SQL-запросов.

- библиотека *utility* – включает полезные инструменты, такие как класс *std::pair* для создания пары значений и функции для работы с ними.

- библиотека *vector* – предоставляет контейнер *std::vector*, представляющий собой динамический массив с удобным интерфейсом для добавления, удаления и доступа к элементам.

Проектирование информационной системы предусматривало её работу в терминале на платформе семейства *Unix*, что обеспечивает стабильную и эффективную работу приложения в средах, поддерживающих соответствующие командные оболочки. Учитывая, что операционные системы *Windows* по умолчанию не включают стандартных инструментов для работы с терминалом *Unix*, использование программы в таких системах возможно только при установке дополнительных программных средств, таких как *Windows Subsystem for Linux (WSL)* или эмуляторов терминала.

Разрабатываемая система имеет разграничение прав доступа между двумя типами пользователей: обычным пользователем и администратором. Каж-

дому пользователю предоставляются различные функции в зависимости от его роли в системе. Обычный пользователь имеет доступ к ограниченному набору функций, связанных с обработкой и управлением билетами. Администратор же, помимо этих возможностей, обладает правами на управление станциями, поездами, вагонами и маршрутами, а также на выполнение более широких операций с билетами.

Обычный пользователь системы может выполнять следующие действия:

- Просмотр списка билетов: пользователь может увидеть все зарегистрированные билеты и информацию о них.
- Создание билета: пользователь может создать новый билет для себя или другого пассажира.
- Удаление билета: пользователь может удалить выбранный билет из системы.

Администратор обладает дополнительными правами, включая все функции обычного пользователя, а также:

- Управление станциями: администратор может добавлять, просматривать и удалять станции.
- Управление поездами: администратор может создавать, просматривать и удалять поезда.
- Управление вагонами: администратор имеет возможность добавлять, просматривать и удалять вагоны в поездах.
- Управление местами: администратор может просматривать и изменять доступные места в поездах.
- Управление маршрутами: администратор может просматривать, добавлять и изменять маршруты.
- Работа с билетами: администратор имеет те же возможности, что и обычный пользователь, но дополнительно может изменять или удалять билеты других пользователей.

Каждая из этих функций организована в виде меню, в котором пользователь или администратор может выбрать нужное действие для управления системой.

2.2 Разработка иерархии классов

Исходя из того, какие действия будет выполнять информационная система и какими данными будет оперировать создаются следующие классы:

- Класс *System* – содержит функции для запуска системы и входа в аккаунт.

- Класс *Database* – содержит функции для работы с базой данных.
- Класс *Entity* – абстрактный класс, содержащий общие функции для работы классов пользователей системы.
- Класс *Admin* – содержит функции для работы с классом *Database*, характерные для администратора.
- Класс *User* – содержит функции для работы с классом *Database*, характерные для пользователя.

Иерархия классов представлена на рисунке 2.1:

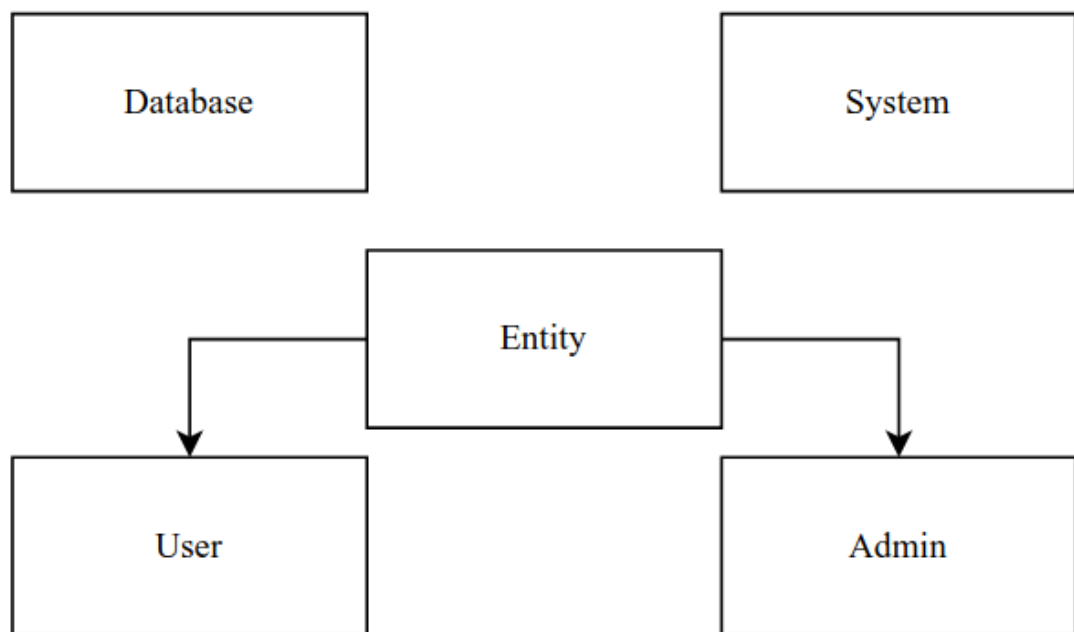


Рисунок 2.1 – Иерархия классов

2.3 Хранение данных

В качестве базы данных будет использоваться *SQLite*. *SQLite* — это встраиваемая база данных, которая не требует отдельного сервера и обеспечивает высокую производительность и простоту использования. Она хранит всю информацию в одном файле и идеально подходит для приложений, которые нуждаются в компактном и эффективном хранилище данных. В связке с *C++* и *CLion* *SQLite* позволяет быстро и легко интегрировать функционал работы с базой данных, что делает её отличным выбором для разработки приложений с базовыми потребностями в хранении данных.

Структура базы данных представлена на рисунке 2.2.

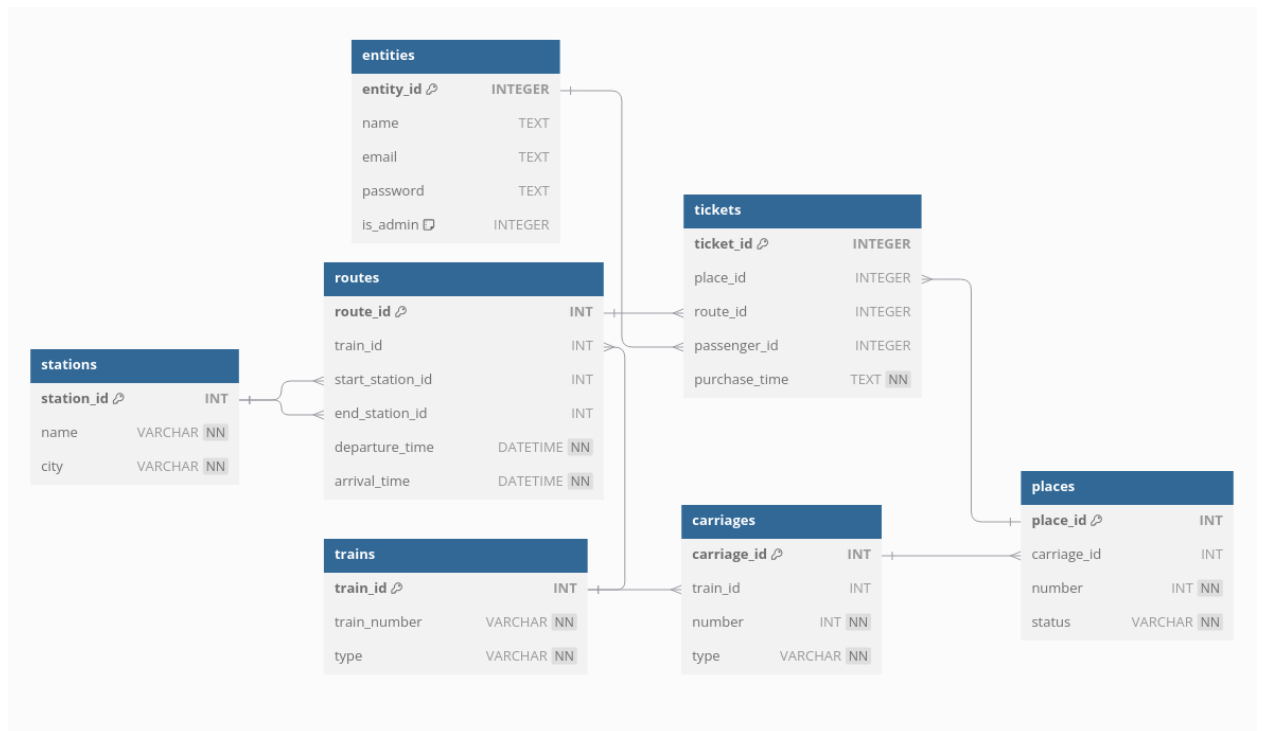


Рисунок 2.2 – Структура базы данных

Структура базы данных организована с использованием нескольких таблиц, каждая из которых отвечает за хранение определенной информации, связанной с вокзалом, поездами, вагонами, местами и билетами.

- Таблица *stations* — содержит информацию о станциях. Каждая станция имеет уникальный идентификатор (*station_id*), название (*name*) и город, в котором она расположена (*city*).

- Таблица *trains* — хранит информацию о поездах. Каждому поезду присваивается уникальный идентификатор (*train_id*), номер поезда (*train_number*) и его тип (*type*).

- Таблица *carriages* — предназначена для хранения информации о вагонах. В каждом вагоне хранится уникальный идентификатор (*carriage_id*), идентификатор поезда, к которому принадлежит вагон (*train_id*), номер вагона (*number*) и его тип (*type*). В таблице также используется внешний ключ, который связывает вагоны с поездами, с условием удаления вагонов при удалении соответствующего поезда.

- Таблица *places* — хранит информацию о местах в вагонах. Каждый элемент в таблице включает уникальный идентификатор места (*place_id*), идентификатор вагона, к которому оно принадлежит (*carriage_id*), номер

места (*number*) и статус места (*status*). Также присутствует внешний ключ, который связывает места с вагонами, с условием удаления мест при удалении соответствующего вагона.

- Таблица *routes* — содержит информацию о маршрутах поездов, включая идентификатор маршрута (*route_id*), идентификатор поезда, который следует по маршруту (*train_id*), идентификаторы начальной и конечной станций (*start_station_id*, *end_station_id*), а также время отправления и прибытия (*departure_time*, *arrival_time*). В таблице используются внешние ключи, связывающие маршруты с поездами и станциями.

- Таблица *entities* — хранит данные о пользователях системы. Каждый пользователь имеет уникальный идентификатор (*entity_id*), имя (*name*), электронную почту (*email*), пароль (*password*) и флаг администратора (*is_admin*).

- Таблица *tickets* — хранит данные о билетах. Каждый билет имеет уникальный идентификатор (*ticket_id*), идентификатор места, на которое был продан билет (*place_id*), идентификатор маршрута, на котором используется билет (*route_id*), идентификатор пассажира (*passenger_id*), который приобрел билет, и время покупки билета (*purchase_time*). В таблице также используются внешние ключи, связывающие билеты с местами, маршрутами и пассажирами.

3 РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

3.1 Описание работы системы

При начальном запуске информационной системы, в терминале появляется меню, представленное на рисунке 3.1

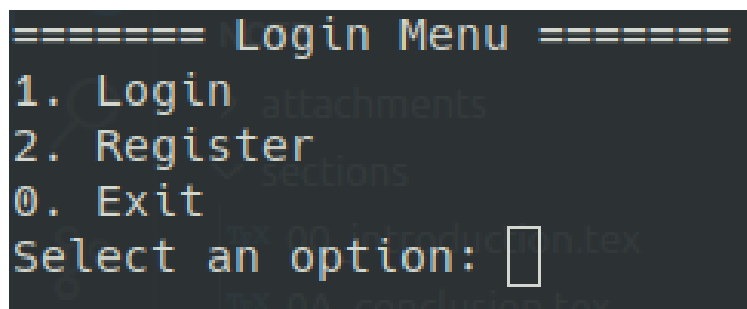


Рисунок 3.1 – Меню входа в систему

Блок-схема данного алгоритма представлена на рисунке 3.2

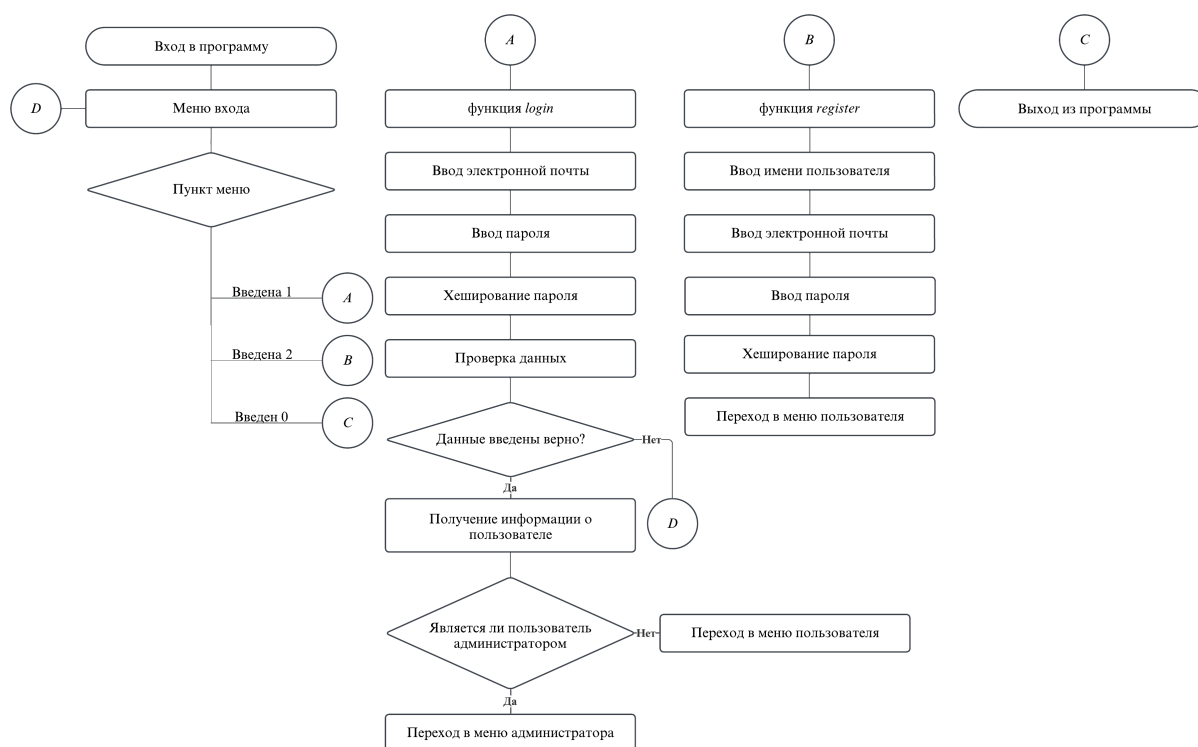


Рисунок 3.2 – Блок-схема алгоритма по входу

Пользователю необходимо выбрать пункт меню. При вводе числа 0 система завершает свою работу (рисунок 3.3)

Для того, чтобы у системы был красивый вид, после ввода каждой команды, терминал очищается.


```
Exiting the system. Goodbye!
```

Рисунок 3.3 – Демонстрация завершения сеанса

Если пользователь введет число 2, система предложит зарегистрироваться. Работа функции для регистрации представлена на рисунке 3.4.

```
Enter your name: Test
Enter your email: test@test.com
Enter your password: testtest
Registration successful!
Press enter to continue...
```

Рисунок 3.4 – Демонстрация регистрации

В системе предусмотрено то, что на одну электронную почту, можно создать только один аккаунт (рисунок 3.5)

```
Enter your name: Test
Enter your email: test@test.com
Enter your password: testtest
Registration failed. Email might already be in use.
Press enter to continue...
```

Рисунок 3.5 – Демонстрация ошибки о том, что почта уже привязана к другому аккаунту

Однако если регистрация пройдет успешно, то пользователь сразу же войдет в систему.

Также, если у пользователя есть аккаунт, он может войти в него, выбрав цифру 1. Для этого необходимо будет ввести *email* и пароль, привязанный к аккаунту.

В системе используется функция для хэширования паролей с использованием алгоритма *SHA-256*, обеспечивающая высокий уровень безопасности. Хэширование представляет собой необратимый процесс преобразования исходного пароля в уникальную последовательность символов фиксированной длины. В данном случае для вычисления хэша применяется библиотека *OpenSSL*. Поэтому пароли в системе хранятся безопасно и, при утечке данных, злоумышленники не смогут восстановить исходные значения паролей, так как процесс хэширования необратим. Даже если хэши будут получены, для нахож-

дения исходного пароля потребуется выполнение вычислительно затратного перебора.

При регистрации, пользователь создается с правами доступа User. Рассмотрим систему со стороны пользователя. Меню выбора, для обычного пользователя представлено на рисунке 3.6

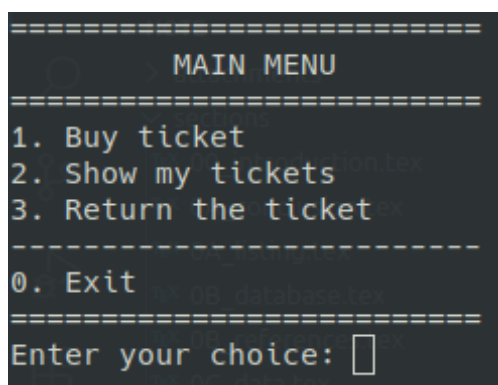


Рисунок 3.6 – Меню выбора действия, для обычного пользователя

По рисунку выше, понятно, что пользователь может проводить манипуляции только со своими билетами. Рассмотрим покупку билета (представлено на рисунке 3.7).

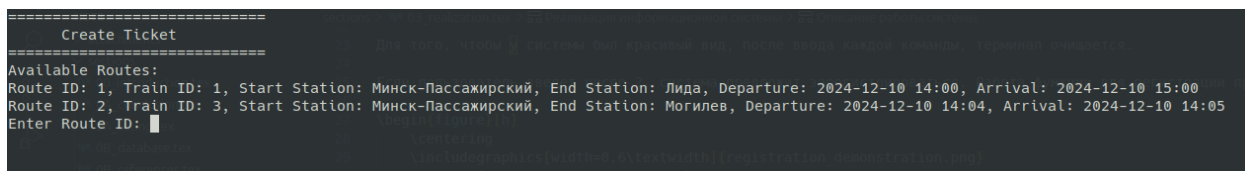


Рисунок 3.7 – Меню покупки билета

Затем, если выбрать корректный поезд, можно выбрать место посадки. В терминал выводятся абсолютно все свободные места (рисунок 3.8).

При этом, дата покупки билета фиксируется как текущая на момент оформления. Это позволяет системе автоматически сохранять временную метку без участия пользователя, обеспечивая точность и достоверность данных. Такое решение упрощает процесс покупки билетов и исключает вероятность ошибок при вводе даты вручную. Данный подход также облегчает последующий анализ данных, например, для статистики покупок или учета занятости мест. Дата сохраняется в таблице в формате *YYYY-MM-DD HH:MM*.

Пользователь может в любое время просмотреть все свои билеты, получив полную информацию о каждом из них. В список включены такие данные, как уникальный идентификатор билета, название маршрута, номер места, дата

```
Enter Route ID: 1
Available Places:
Place ID: 1, Number: 1
Place ID: 2, Number: 2
Place ID: 3, Number: 3
Place ID: 4, Number: 4
Place ID: 5, Number: 5
Place ID: 6, Number: 6
Place ID: 7, Number: 7
Place ID: 8, Number: 8
Place ID: 9, Number: 9
Place ID: 10, Number: 10
Place ID: 11, Number: 11
Place ID: 12, Number: 12
Place ID: 13, Number: 13
Place ID: 14, Number: 14
Place ID: 15, Number: 15
Place ID: 16, Number: 16
Place ID: 17, Number: 17
Place ID: 18, Number: 18
Place ID: 19, Number: 19
Place ID: 20, Number: 20
Place ID: 21, Number: 21
Place ID: 22, Number: 22
Place ID: 23, Number: 23
Place ID: 24, Number: 24
Place ID: 25, Number: 25
Place ID: 26, Number: 26
Place ID: 27, Number: 27
Place ID: 28, Number: 28
Place ID: 29, Number: 29
Place ID: 30, Number: 30
Place ID: 31, Number: 1
Place ID: 32, Number: 2
Place ID: 33, Number: 3
Place ID: 34, Number: 4
Place ID: 35, Number: 5
Place ID: 36, Number: 6
Place ID: 37, Number: 7
Place ID: 38, Number: 8
Place ID: 39, Number: 9
Place ID: 40, Number: 10
Place ID: 41, Number: 11
Place ID: 42, Number: 12
```

Рисунок 3.8 – Фрагмент меню выбора места для посадки

и время отправления, а также место назначения. Это представлено наглядно в пользовательском интерфейсе (рисунок 3.9).

Кроме того, пользователь имеет возможность вернуть приобретённый билет. Возврат может быть осуществлён через интерфейс системы, где отображаются доступные для возврата билеты. Данный функционал позволяет пользователю гибко управлять своими поездками, например, изменяя планы или освобождая место для другого пассажира. При возврате билета данные обновляются в системе в реальном времени, что обеспечивает актуальность информации о доступных местах (рисунок 3.10).

Меню администратора в информационной системе вокзала предоставляет возможности для управления всеми ключевыми элементами инфраструктуры. Администратор может добавлять, редактировать и удалять поезда, указывая информацию о номере, маршруте, времени отправления и количестве вагонов. Для каждого поезда можно управлять вагонами, редактировать их

```
=====
Show Tickets
=====
Your Tickets:
Ticket ID: 3, Route: 1, Place: 59, Purchase Time: 2024-12-11 17:30
Press enter to continue...|
```

Рисунок 3.9 – Меню просмотра всех билетов пользователя

```
=====
Delete Ticket
=====
Your Tickets:
Ticket ID: 3, Route: 1, Place: 59, Purchase Time: 2024-12-11 17:30
Enter Ticket ID to delete: 3
Ticket deleted successfully.
Press enter to continue...|
```

Рисунок 3.10 – Меню возврата билетов

параметры, включая типы и количество мест. Система позволяет управлять билетами, включая создание, изменение и удаление бронирований, а также назначать их на конкретные места в вагонах. Администратор может управлять станциями, путями и пользователями, обновлять их данные, а также следить за состоянием доступных мест и путевых маршрутов.

Рассмотрим некоторые ключевые функции, из тех, что перечислены выше. Каждая сущность в базе данных имеет полный набор *CRUD*-запросов.

CRUD — это аббревиатура, обозначающая четыре основные операции, используемые для управления данными в базе данных: *Create* (создание), *Read* (чтение), *Update* (обновление) и *Delete* (удаление). Эти операции предоставляют полный набор инструментов для выполнения всех необходимых действий с сущностями в системе.

Система проверяет значение поля *is_admin* в таблице *entities* при входе пользователя, чтобы определить его уровень доступа. Если значение равно *true*, пользователь получает права администратора и доступ к функциональности управления ключевыми объектами системы, такими как поезда, вагоны, места, билеты, пути, станции и пользователи. В противном случае, пользователь рассматривается как обычный клиент с доступом только к покупке и просмотру билетов. Добавление нового администратора возможно только вручную через прямое изменение записей в базе данных, что обеспечивает дополнительный уровень безопасности и контроля.

Меню администратора представлено на рисунке 3.11.

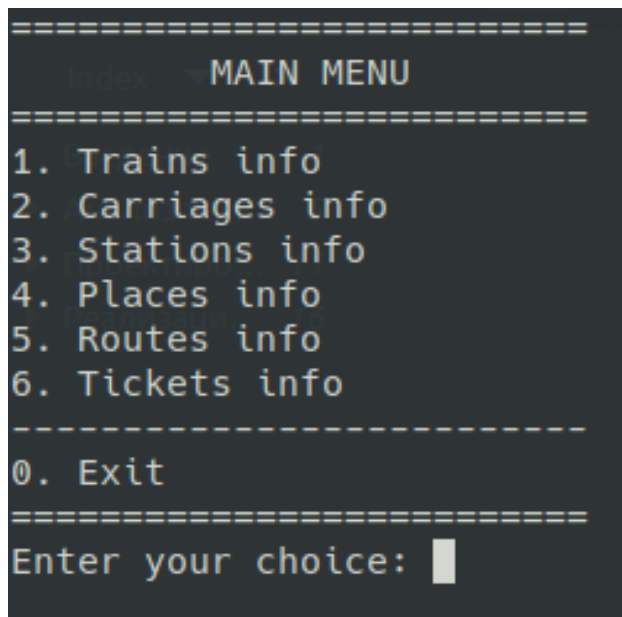


Рисунок 3.11 – Меню администратора

В нем, так же как и в меню обычного пользователя, можно выбирать различные пункты меню. Рассмотрим ключевые функции некоторых пунктов меню.

Меню поездов представлено на рисунке 3.12. Информационная система предоставляет функции для добавления, удаления и просмотра поездов.

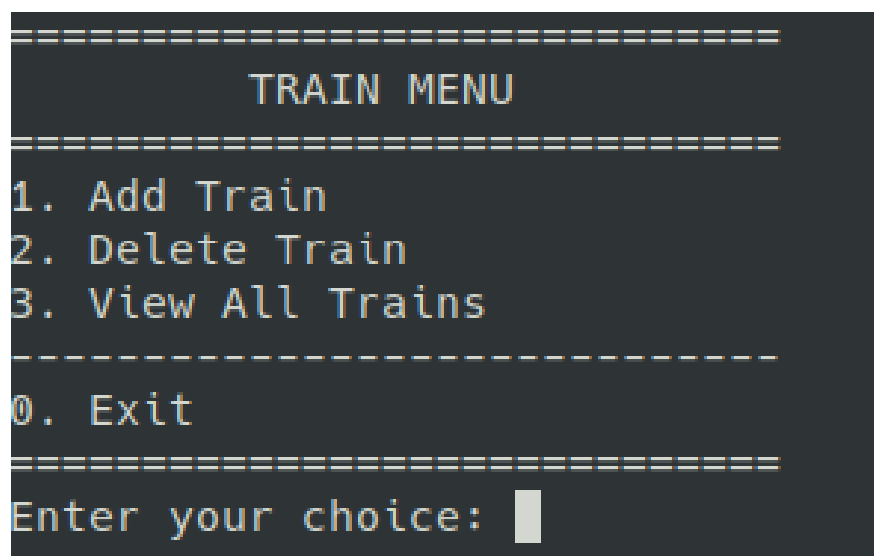


Рисунок 3.12 – Меню поездов

Рассмотрим добавление поезда. Для добавления поезда необходимо ввести его номер, а также тип поезда (рисунок 3.13).

Также, рассмотрим удаление поезда по его идентификатору. При попытке удаления поезда, система выводит список всех доступных поездов. Удалим

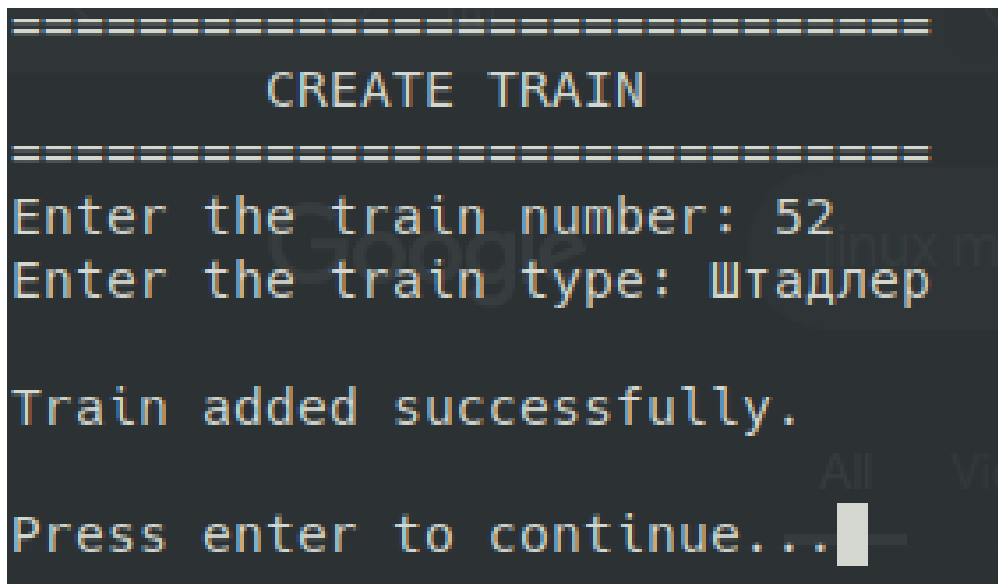


Рисунок 3.13 – Функционал по добавлению поезда в базу данных

поезд номер 7, а затем попробуем прочитать список всех доступных поездов (рисунок 3.14 и рисунок 3.15)

При удалении поезда по его идентификатору система выполняет несколько последовательных шагов для обеспечения корректности операций.

Во-первых, при запросе на удаление система выводит пользователю актуальный список всех доступных поездов, чтобы он мог визуально подтвердить данные о поезде, который требуется удалить. Это особенно важно для предотвращения ошибок, связанных с удалением неправильного объекта.

Например, если пользователь выберет удаление поезда, идентификатор которого равен 7, система проверит его наличие в базе данных. Если поезд с таким идентификатором существует, он будет удален. При этом система также автоматически удаляет связанные с этим поездом данные, такие как вагоны, места и билеты, чтобы избежать несоответствий в базе данных.

После выполнения операции удаления пользователь может запросить обновленный список доступных поездов. Система вновь выполнит запрос к базе данных и отобразит актуальную информацию, из которой теперь будет исключен поезд с идентификатором 7. Это наглядно продемонстрировано на рисунках 3.14 и 3.15, где показано состояние данных до и после удаления поезда.

На рисунке 3.15 видно, что поезда с идентификатором 7 больше нету в базе данных.

Теперь рассмотрим операции над вагонами. По рисунку 2.2 видно, что

```
=====
                        TRAINS LIST
=====

Trains:
- ID: 1, Номер поезда: 001, Тип: Штадлер
- ID: 7, Номер поезда: 2, Тип: Штадлер
=====

=====
                        DELETE TRAIN
=====

Enter the train ID to delete: 7
```

Рисунок 3.14 – Удаление поезда с идентификатором 7

у таблицы *carriage* есть зависимость от *trains*. Также, в каждом вагоне есть места. Рассмотрим создание нового вагона (рисунок 3.16). При создании вагона, система выполняет несколько последовательных операций: создание записи в таблице *carriage*, а также наполнение его местами. Это сделано для того, чтобы упростить задачу по созданию мест. В каждом вагоне может быть от 30 мест. Программа создает 30 мест. Чуть позже, с помощью меню по управлению местами, администратор может корректировать их количество.

От администратора система требует идентификатор существующего поезда. При этом, система показывает список всех поездов, а если их нет, то система оповестит об этом администратора. Затем система требует номер вагона и его тип.

Не будем рассматривать как удаляются вагоны. Все выглядит так же, как и с поездами. Аналогично с просмотром всех доступных вагонов.

Рассмотрим меню по управлению путями (рисунок 3.17).

```
=====
                TRAINS LIST
=====

Trains:
- ID: 1, Номер поезда: 001, Тип: Штадлер
=====

Press enter to continue...|
```

Рисунок 3.15 – Список всех поездов после удаления поезда с идентификатором 7

```
=====
                CREATE CARRIAGE
=====
Choose train ID:
Train ID: 1
Enter train ID: 1
Enter carriage number: 02
Enter carriage type (Compartment, Economy, Luxury): Economy

Carriage added successfully.
```

Рисунок 3.16 – Функционал по созданию вагона

```
=====
                Manage Routes
=====
1. Create Route
2. Delete Route
3. View Routes
0. Back
=====
Enter your choice: |
```

Рисунок 3.17 – Меню управления поездками

При разработке информационной системы для вокзала функции удаления и просмотра различных сущностей реализуются по аналогичному принципу, что и для других элементов системы, таких как поезда, вагоны, станции и билеты. Эти действия требуют простого выполнения запросов к базе данных, что позволяет быстро и эффективно обрабатывать соответствующие запросы. В связи с этим, более детальное рассмотрение их реализации в данном разделе нецелесообразно. Вместо этого акцент далее сделан на рассмотрении процесса создания сущностей, поскольку это сложная и важная часть функционала системы.

Поездка в данной информационной системе является составной сущностью, включающей поезд, вагоны и две станции — станцию отправления и станцию прибытия. При создании новой поездки требуется указать существующий поезд из базы данных, что обеспечивает связь поездки с реальными данными о составе. Также указываются станции отправления и прибытия, которые должны быть предварительно внесены в базу данных. Для обеспечения корректности планирования необходимо задать время отправления и прибытия, что позволяет учитывать расписание движения поездов и гарантировать точность работы системы (рисунок 3.18).

```
Available Trains:
ID: 1, Номер поезда: 001, Тип: Штадлер

Available Stations:
ID: 1, Название: Минск-Пассажирский (Минск)
ID: 2, Название: Гомель (Гомель)
ID: 3, Название: Брест (Брест)
ID: 4, Название: Витебск (Витебск)
ID: 5, Название: Гродно (Гродно)
ID: 6, Название: Могилев (Могилев)
ID: 7, Название: Орша-Центральная (Орша)
ID: 8, Название: Барановичи-Центральные (Барановичи)
ID: 9, Название: Полоцк (Полоцк)
ID: 10, Название: Лида (Лида)
ID: 11, Название: Уручье (Минск)

Enter Train ID: 1
Enter Start Station ID: 1
Enter End Station ID: 2
Enter Departure Time (YYYY-MM-DD HH:MM): 2024-12-31 19:00
Enter Arrival Time (YYYY-MM-DD HH:MM): 2024-12-31 21:50
Route created successfully.

Press enter to continue...
```

Рисунок 3.18 – Функционал по созданию поездки

Рассмотрим удаление поездки. Удаление поездки представлено на рисунке 3.19

```

=====
Available Routes:
=====
Route ID: 3, Train: 001, Start Station: Минск-Пассажирский, End Station: Гомель, Departure: 2024-12-31 19:00, Arrival: 2024-12-31 21:50
Enter Route ID to delete: 3
Route deleted successfully.
Press enter to continue...

```

Рисунок 3.19 – Функционал по удалению поездки

От администратора система требует идентификатор существующей поездки. При этом, система показывает список всех поездок, а если их нет, то система оповестит об этом администратора(рисунок 3.20).

```

=====
No routes found.
=====
Press enter to continue...

```

Рисунок 3.20 – Оповещение системы о том, что поездки отсутствуют

Рассмотрим меню билетов(рисунок 3.21)

```

=====
Ticket Management
=====
1. Create Ticket
2. View Tickets
3. Delete Ticket
0. Back to Main Menu
=====
Enter your choice:

```

Рисунок 3.21 – Меню билетов

Рассмотрим функцию создания билета(рис 3.22 и 3.23).

```
=====
Create Ticket
=====
Available Routes:
Route ID: 2, Train ID: 3, Start Station: Минск-Пассажирский, End Station: Могилев, Departure: 2024-12-10 14:04, Arrival: 2024-12-10 14:05
Route ID: 4, Train ID: 8, Start Station: Минск-Пассажирский, End Station: Гомель, Departure: 2025-01-01 01:01, Arrival: 2025-01-01 13:00
Enter Route ID: 4
Available Places:
Place ID: 121, Number: 1
Place ID: 122, Number: 2
Place ID: 123, Number: 3
```

Рисунок 3.22 – Функционал по созданию билета ч.1

```
Enter Place ID: 140
Available Passengers:
ID: 1, Name: Stanislav, Email: s.markovtsev@gmail.com
ID: 2, Name: s, Email: s
ID: 3, Name: se, Email: se
ID: 4, Name: ef, Email: ef
ID: 5, Name: ser, Email: ser
ID: 6, Name: ter, Email: ter
ID: 7, Name: ds, Email: ds
ID: 8, Name: adlm, Email: adl
ID: 9, Name: Mawkovtsev, Email: s.markovtsev
ID: 10, Name: Markovtsev, Email: s.gmail
ID: 11, Name: admin, Email: admin
ID: 12, Name: sonya, Email: kalik@gmail.fimoz
ID: 13, Name: user, Email: user
ID: 14, Name: usr, Email: usr
ID: 15, Name: Stanislav, Email: example@example.com
ID: 16, Name: Test, Email: test@test.com
Enter Passenger ID: 1
Enter Purchase Time (YYYY-MM-DD HH:MM): 2024-12-12
Ticket created successfully.
```

Рисунок 3.23 – Функционал по созданию билета ч.2

В отличие от функции покупки билета в меню пользователя(3.7), в создании билета в меню администратора пассажир, а так же дата покупки задаются вручную.

Также в системе есть вспомогательные функции:

- функция *handleInvalidInput()* – функция вызывается после каждого пользовательского ввода. Она нужна для того, чтобы после некорректного ввода, программа не завершала свою работу.
- функция *clearScreen()* – функция вызывается перед каждым пользовательским вводом. Она нужна для того, чтобы интерфейс создавал впечатление обновления экрана.

- функция *pressToContinue()* – функция вызывается после завершения какого то блока программы, например перед *break* или *return*.
- функция *hashPassword()* – функция используется для безопасного хранения паролей в базе данных.
- функция *getCurrentTime* – функция используется для получения текущего времени. Например, когда пользователь оформляет билет.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был разработан проект реляционной базы данных для информационной системы вокзала, направленной на автоматизацию ключевых процессов.

В результате анализа предметной области были определены основные сущности и взаимосвязи между ними, что позволило создать структуру базы данных, включающую таблицы для управления информацией о станциях, маршрутах, поездах, вагонах, местах, билетах и пользователях. Каждая таблица реализует свои уникальные функции: хранение данных о рейсах, статусах мест, расписаниях, продаже билетов и данных пользователей.

Для взаимодействия с каждой таблицей разработаны функции создания, удаления и отображения данных. Это обеспечивает удобное управление базой данных через консольное приложение. В рамках системы реализовано разграничение прав доступа, что позволяет разделять пользователей на обычных и администраторов. Администраторы имеют расширенные возможности, такие как управление поездами, вагонами, расписанием и пользователями, тогда как обычные пользователи могут, например, просматривать расписание и приобретать билеты.

В ходе реализации программы использовались подходы объектно-ориентированного программирования, включая иерархию классов, детально описанную с определением их полей и методов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Павловская, Т. А. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. — СПб.: Питер, 2010. — 461 с.
- [2] Лафоре, Р. Объектно-ориентированное программирование в C++/ Р. Лафоре – СПб: Питер Ком, 2019. – 923 с.
- [3] Вендров, А.М. Практикум по проектированию программного обеспечения экономических информационных систем : учебное пособие / А.М. Вендров. – Москва : Финансы и статистика, 2004. – 192 с. : ил. – (UML CASE).
- [4] Шилдт, Г. Самоучитель C++: Пер. с англ. – 3-е изд. – СПб.: БХВ-Петербург, 2003. — 688 с.
- [5] Страуструп, Б.Б Язык программирования C++/ Б. Страуструп – М.: Бином, 2022. – 369 с.
- [6] Вайсфельд, М. Объектно-ориентированное мышление. – СПб: Питер, 2014. – 27с.
- [7] СТП 01-2017. Стандарт предприятия. Дипломные проекты (работы). Общие требования. - Минск: БГУИР, 2017. - 169 с.
- [8] *Drawio* [Электронный ресурс]. – Режим доступа: <https://app.diagrams.net/>
- [9] БЧ. Мой поезд [Электронный ресурс]. – Режим доступа: <https://pass.rw.by/ru/>

ПРИЛОЖЕНИЕ А
(обязательное)
Фрагмент исходного кода приложения

```
#include "../Admin.h"

#include "../Utils.h"

Admin::Admin(std::string name, int userId) {
    this->name = name;
    this->userId = userId;
}

void Admin::start() {
    std::cout << "Hi, " << name << "!" << std::endl;

    if (!db.openDatabase(dbName)) {
        std::cerr << "Failed to open the database. Terminating." << std::endl;
        std::exit(EXIT_FAILURE);
    }

    clearScreen();

    while (isWork) {
        std::cout << "===== " <<
std::endl;
        std::cout << "    MAIN MENU    " << std::endl;
        std::cout << "===== " <<
std::endl;
        std::cout << "1. Trains info" << std::endl;
        std::cout << "2. Carriages info" << std::endl;
        std::cout << "3. Stations info" << std::endl;
        std::cout << "4. Places info" << std::endl;
        std::cout << "5. Routes info" << std::endl;
        std::cout << "6. Tickets info" << std::endl;
        std::cout << "-----" << std::endl;
        std::cout << "0. Exit" << std::endl;
```

```

std::cout << "===== " <<
std::endl;
std::cout << "Enter your choice: ";

int command;
std::cin >> command;
handleInvalidInput();

switch (command) {
    case 1:
        clearScreen();
        std::cout << "\n===== TRAINS INFO =====\n" << std::endl;
        printTrainsMenu();
        break;

    case 2:
        clearScreen();
        std::cout << "\n===== CARRIAGES INFO =====\n" <<
std::endl;
        printCarriagesMenu();
        break;

    case 3:
        clearScreen();
        std::cout << "\n===== STATIONS INFO =====\n" <<
std::endl;
        printStationsMenu();
        break;

    case 4:
        clearScreen();
        std::cout << "\n===== PLACES INFO =====\n" << std::endl;
        printPlacesMenu();
        break;

    case 5:
        clearScreen();
        std::cout << "\n===== ROUTES INFO =====\n" << std::endl;

```



```

        printRoutesMenu();
        break;

    case 6:
        clearScreen();
        std::cout << "\n===== TICKETS INFO =====\n" << std::endl;
        printTicketsMenu();
        break;

    case 0:
        clearScreen();
        std::cout << "\nExiting... Goodbye!\n" << std::endl;
        exit();
        break;

    default:
        clearScreen();
        std::cout << "\nInvalid choice. Please try again.\n" << std::endl;

        pressToContinue();
        break;
    }
}
}

```

```

void Admin::createStation() {
    clearScreen();
    std::string name, city;

    std::cout << "===== " <<
std::endl;
    std::cout << "    CREATE STATION    " << std::endl;
    std::cout << "===== " <<
std::endl;

    std::cout << "Enter the station name: ";
    std::cin.ignore();
    std::getline(std::cin, name);
}

```

```

std::cout << "Enter the city: ";
std::getline(std::cin, city);

if (db.createLocation(name, city) == SQLITE_OK) {
    std::cout << "\nStation added successfully.\n";
} else {
    std::cerr << "\nError adding the station.\n";
}

pressToContinue();
}

void Admin::readStations() {
    clearScreen();

    std::cout << "===== " <<
std::endl;
    std::cout << "      STATIONS LIST      " << std::endl;
    std::cout << "===== " <<
std::endl;

    std::vector<std::string> stations = db.readLocations();

    if (stations.empty()) {
        std::cout << "\nNo stations in the database.\n";
    } else {
        std::cout << "\nStations:\n";
        for (const auto &station: stations) {
            std::cout << " - " << station << std::endl;
        }
    }
    std::cout << "=====\n" <<
std::endl;

    pressToContinue();
}

```

```

void Admin::deleteStation() {
    clearScreen();
    readStations();

    int station_id;
    std::cout << "===== " <<
std::endl;
    std::cout << "    DELETE STATION    " << std::endl;
    std::cout << "===== " <<
std::endl;

    std::cout << "Enter the station ID to delete: ";
    std::cin >> station_id;

    if (db.deleteLocation(station_id) == SQLITE_OK) {
        std::cout << "\nStation deleted successfully.\n";
    } else {
        std::cout << "\nError deleting the station.\n";
    }

    pressToContinue();
}

void Admin::printStationsMenu() {
    clearScreen();

    std::cout << "===== " <<
std::endl;
    std::cout << "    STATIONS MENU    " << std::endl;
    std::cout << "===== " <<
std::endl;
    std::cout << "1. Add a station" << std::endl;
    std::cout << "2. Delete station" << std::endl;
    std::cout << "3. View all stations" << std::endl;
    std::cout << "0. Exit" << std::endl;

    int command;
    std::cout << "Enter your choice: ";

```

```

std::cin >> command;
handleInvalidInput();

switch (command) {
    case 1: {
        clearScreen();
        createStation();
        break;
    }
    case 2: {
        clearScreen();
        deleteStation();
        break;
    }
    case 3: {
        clearScreen();
        readStations();
        break;
    }
    case 0: {
        clearScreen();
        return;
    }
    default: {
        clearScreen();
        std::cout << "\nInvalid choice. Please try again.\n";
        pressToContinue();
        break;
    }
}

void Admin::createTrain() {
    clearScreen();

    std::string train_number, type;
    std::cout << "===== " <<
std::endl;
    std::cout << "    CREATE TRAIN    " << std::endl;

```

```

        std::cout << "===== " <<
std::endl;

        std::cout << "Enter the train number: ";
        std::cin.ignore();
        std::getline(std::cin, train_number);

        std::cout << "Enter the train type: ";
        std::getline(std::cin, type);

        if (db.createTrain(train_number, type) == SQLITE_OK) {
            std::cout << "\nTrain added successfully.\n";
        } else {
            std::cout << "\nError adding the train.\n";
        }

        pressToContinue();
    }

    void Admin::deleteTrain() {
        clearScreen();

        readTrains();
        int train_id;
        std::cout << "===== " <<
std::endl;
        std::cout << "      DELETE TRAIN      " << std::endl;
        std::cout << "===== " <<
std::endl;

        std::cout << "Enter the train ID to delete: ";
        std::cin >> train_id;

        if (db.deleteTrain(train_id) == SQLITE_OK) {
            std::cout << "\nTrain deleted successfully.\n";
        } else {
            std::cout << "\nError deleting the train.\n";
        }
    }

```

```

        pressToContinue();
    }

void Admin::readTrains() {
    clearScreen();

    std::vector<std::string> trains = db.readTrains();

    std::cout << "===== " <<
std::endl;
    std::cout << "    TRAINS LIST    " << std::endl;
    std::cout << "===== " <<
std::endl;

    if (trains.empty()) {
        std::cout << "\nNo trains in the database.\n";
    } else {
        std::cout << "\nTrains:\n";
        for (const auto &train: trains) {
            std::cout << " - " << train << std::endl;
        }
    }
    std::cout << "===== \n" <<
std::endl;
}

void Admin::printTrainsMenu() {
    clearScreen();

    std::cout << "===== " <<
std::endl;
    std::cout << "    TRAIN MENU    " << std::endl;
    std::cout << "===== " <<
std::endl;
    std::cout << "1. Add Train    " << std::endl;
    std::cout << "2. Delete Train " << std::endl;

```

```

std::cout << "3. View All Trains      " << std::endl;
std::cout << "-----" << std::endl;
std::cout << "0. Exit                " << std::endl;
std::cout << "===== " <<
std::endl;
std::cout << "Enter your choice: ";

int command;
std::cin >> command;
handleInvalidInput();
std::cin.ignore();
switch (command) {
    case 1: {
        createTrain();
        break;
    }
    case 2: {
        deleteTrain();
        break;
    }
    case 3: {
        readTrains();
        pressToContinue();
        break;
    }
    case 0: {
        clearScreen();
        return;
    }
    default: {
        clearScreen();
        std::cout << "\nInvalid choice. Please try again.\n";

        pressToContinue();
        break;
    }
}
}

```

```

void Admin::createCarriage() {
    clearScreen();

    std::cout << "===== " <<
std::endl;
    std::cout << "    CREATE CARRIAGE    " << std::endl;
    std::cout << "===== " <<
std::endl;

    std::cout << "Choose train ID: " << std::endl;
    std::vector<int> train_ids = db.getTrainIds();
    if (train_ids.empty()) {
        std::cout << "No trains available.\n";
        return;
    }

    for (int id: train_ids) {
        std::cout << "Train ID: " << id << std::endl;
    }

    int train_id;
    std::cout << "Enter train ID: ";
    std::cin >> train_id;

    if (std::cin.fail()) {
        std::cout << "Invalid input for train ID.\n";
        return;
    }

    auto it = std::find(train_ids.begin(), train_ids.end(), train_id);
    if (it == train_ids.end()) {
        std::cout << "Invalid train ID.\n";
        return;
    }

    int number;
    std::string type;

```



```

std::cout << "Enter carriage number: ";
std::cin >> number;
std::cout << "Enter carriage type (Compartment, Economy, Luxury): ";
std::cin >> type;

if (db.createCarriage(train_id, number, type) == SQLITE_OK) {
    std::cout << "\nCarriage added successfully.\n";
} else {
    std::cout << "\nError adding carriage.\n";
}

pressToContinue();
}

void Admin::deleteCarriage() {
    clearScreen();

    int carriage_id;
    std::cout << "===== " <<
std::endl;
    std::cout << "      DELETE CARRIAGE      " << std::endl;
    std::cout << "===== " <<
std::endl;

    std::cout << "Enter carriage ID to delete: ";
    std::cin >> carriage_id;

    if (db.deleteCarriage(carriage_id) == SQLITE_OK) {
        std::cout << "\nCarriage deleted successfully.\n";
    } else {
        std::cout << "\nError deleting carriage.\n";
    }

    pressToContinue();
}

```

```

void Admin::readCarriages() {
    clearScreen();

    std::vector<std::string> carriages = db.readCarriages();

    std::cout << "===== " <<
std::endl;
    std::cout << "    CARRIAGES LIST    " << std::endl;
    std::cout << "===== " <<
std::endl;

    if (carriages.empty()) {
        std::cout << "\nNo carriages available.\n";
    } else {
        std::cout << "\nCarriages:\n";
        for (const auto &carriage: carriages) {
            std::cout << " - " << carriage << std::endl;
        }
    }

    std::cout << "===== " <<
std::endl;

    pressToContinue();
}

void Admin::printCarriagesMenu() {
    clearScreen();

    std::cout << "===== " <<
std::endl;
    std::cout << "    CARRIAGES MENU    " << std::endl;
    std::cout << "===== " <<
std::endl;
    std::cout << "1. Add carriage        " << std::endl;
    std::cout << "2. Delete carriage     " << std::endl;
    std::cout << "3. View all carriages  " << std::endl;
    std::cout << "4. Exit                " << std::endl;

```

```

        std::cout << "===== " <<
std::endl;
        std::cout << "Enter your choice: ";

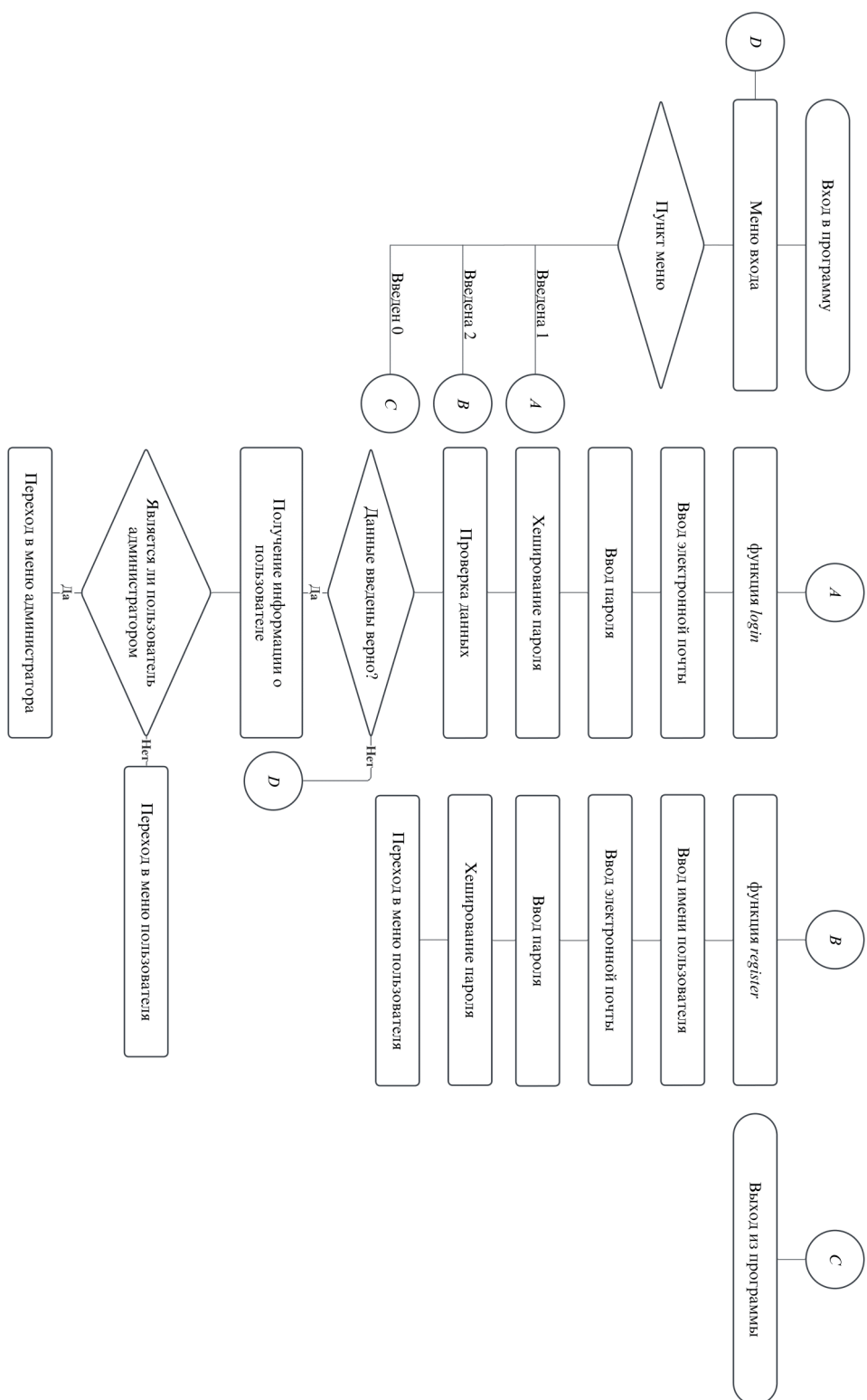
        int command;
        std::cin >> command;
        handleInvalidInput();
        std::cin.ignore();
        switch (command) {
            case 1: {
                createCarriage();
                break;
            }
            case 2: {
                deleteCarriage();
                break;
            }
            case 3: {
                readCarriages();
                break;
            }
            case 4: {
                return;
            }
            default: {
                clearScreen();
                std::cout << "\nInvalid choice. Please try again.\n";
                pressToContinue();
                break;
            }
        }
    }
}

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Блок-схема алгоритма входа в приложение



ГЧИР.320602.001 СА					Блок-схема алгоритма входа в приложение			Лит.	Масса	Масштаб
Изм.	Лист	№ докум.	Подпись	Дата						
Разраб.		Марковцев								
Провер.		Ючков								
Т. Контр.										
Реценз.										
Н. Контр.								Лист 45	Листов 46	
Утверд.								ИТАС, гр. 320602		

Обозначение					Наименование	Дополнительные сведения			
					<u>Текстовые документы</u>				
БГУИР КР 1-53-01-02 219 ПЗ					Пояснительная записка	30 с.			
					<u>Графические документы</u>				
ГУИР.320602.001 СА					Блок схема алгоритма входа в приложение	Формат А4			
					</				