Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Программирование сетевых приложений»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  старший преподаватель  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.М. Унучек |
|  | \_\_\_\_\_\_\_.2021 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«**Программная поддержка деятельности компании по предоставлению туристических услуг**»**

БГУИР КП 1-40 05 01-09 051 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 910101  Атаев Исмайыл Мерданович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_\_\_\_\_.2021  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2021

**РЕФЕРАТ**

БГУИР КП 1-40 05 01-10 009 ПЗ

Атаев И.М. Программная поддержка деятельности компании по предоставлению туристических услуг / И. Атаев. – Минск: БГУИР, 2021.

Пояснительная записка 65 стр., 28 рис., 14 источников, 13 приложения.

ПРИЛОЖЕНИЕ, ТУРИСТИЧЕСКОЙ ФИРМЫ, ДЕСКТОП ПРИЛОЖЕНИЯ, АДИНИМТРАТОР, ИНТЕРНЕТ, ТУР, КАТАЛОГ, ПОЛЬЗОВАТЕЛЬ, КЛИЕНТ, СЕРВЕР, КОНТРОЛЛЕР, МОДЕЛЬ, ВИД, ИНТЕРФЕЙС, ПАТТЕРН, КОНСТРУКЦИЯ, ШАБЛОН, ПРИЛОЖЕНИЕ.

*Цель проектирования*: разработка десктоп приложения для интернет-магазина для удобной приобретение билета туров.

*Методология проведения работы*: в процессе решения поставленных задач были использованы принципы и методы паттерна MVC (Модель-Представление-Контроллер), методы защиты приложения от посторонних пользователей, были использованы конфигурации платформы для авторизации и работы с данными.

*Результаты работы*: выполнена обработка процедура администраторами и клиентом добавление туров, другими администраторами, было обработано бронирование туров для клиентов, добавлено процедура отслеживание стадию заказа со стороны пользователей и администратора, создан поисковик, позволяющий находить клиента, также туров, который обрабатывает полученные администратором данные через сопоставление, введенных в поисковик. Добавлено редактирование туров пользователей по всем полям. Также создана возможность отмены заказа. Уделено внимание технико-экономическим вопросам данного проекта, который представляет в виде статистики продаж туров для клиентов.

*Область применения результатов*: могут быть использованы при проектировании десктоп приложение с удобным интерфейс для обработки процедур продажи и туров.

**Содержание**

[Введение](#_Toc58827281) 3

[1 Оисания предметной области](#_Toc58827282) 4

[1.1 Сущность и функция Туристической фирмы](#_Toc58827283) 4

[2 Постановка задачи и обзор методов её решения](#_Toc58827285) 5

[2.1 Постановка задачи и обзор методов её решения](#_Toc58827286) 5

2.2 Диаграмма вариантов использования UML........................................................................9

[2.3 Модели представления системы и их описание](#_Toc58827292) 11

[2.4 Применение паттернов проектирования](#_Toc58827293) 12

[3 Описание основного процесса функционирования работы системы управления на основе стандарта idef0](#_Toc58827294) 15

[4 Информационная модель системы управление](#_Toc58827295) 16

[4.1 Описание информационной модели](#_Toc58827296) 16

[5 Руководство пользователя](#_Toc58827297) 18

[5.1 Руководство серверной части приложения](#_Toc58827298) 18

[5.2 Спецификация вариантов использования системы](#_Toc58827299) 18

[6 Описание алгоритмов, реализующих бизнес-логику работы системы](#_Toc58827303) 19

[6.1 Алгоритм работы всей программы](#_Toc58827304) 19

[6.2 Алгоритм работы функции авторизации](#_Toc58827305) 19

[6.3 Руководство клиентской части приложения](#_Toc58827306) 20

[6.4 Руководство администраторной части приложения](#_Toc58827306) 26

[Заключение](#_Toc58827307) 33

[Список использованных источников](#_Toc58827308) 35

[Приложение А (обязательное)](#_Toc58827309) 38

[Приложение Б (обязательное)](#_Toc58827310) 41

[Приложение В (обязательное)](#_Toc58827311) 41

[Приложение Г (обязательное)](#_Toc58827312) 42

[Приложение Д (обязательное)](#_Toc58827313) 43

[Приложение Е (обязательное)](#_Toc58827314) 47

[Приложение Ж (обязательное)](#_Toc58827315) 48

[Приложение З (обязательное)](#_Toc58827315) 50

[Приложение И (обязательное)](#_Toc58827315) 67

[Приложение К (обязательное)](#_Toc58827315) 68

[Приложение Л (обязательное)](#_Toc58827315) 69

[Приложение М (обязательное)](#_Toc58827315) 70

[Приложение Н (обязательное)](#_Toc58827315) 71

**Введение**

Среди многообразия поисков путей развития рынка, средств производства, новых направлений деятельности коммерческо-посреднических организаций и предприятий вызывают значительный интерес научные исследования и практические новации. В течение последних лет бурно развиваются основанные на информатике новые технологии.

Актуальность темы заключается в том, что информационные системы (ИС) занимают в этих технологиях центральное положение.

Объектом автоматизации является туристическая фирма. ИС являются одним из важных источников минимизации издержек и затрат на предприятии и оптимизации методов ведения бизнеса. Для проектирования эффективной ИС предприятий исследуются и определяются: задачи, источники информации и информационная база, необходимые для качественного расчета экономических, технических и хозяйственных показателей предприятия; стратегия функционирования информационных систем; основные общесистемные принципы, необходимые при создании ИС; модели стратегии создания и развития ИС; топология и сетевые операционные системы, используемые для реализации ИС.

В мире, который нас окружает, циркулируют огромные потоки информации. И со временем они только увеличиваются. Поэтому для любого предприятия актуальна проблема управления данными и информационными потоками. В основном используют базы данных, которые позволяют наиболее эффективно хранить, систематизировать и структурировать большие объемы информации. На сегодняшний день нельзя представить функционирование большинства крупных и мелких предприятий различных отраслей.

Существует множество причин для приведения информации к цифровому виду. На данный момент стоимость сохранения информации на цифровых носителях гораздо меньше, чем на бумаге. С помощью баз данных можно сохранять, структурировать и извлекать информацию самым оптимальным для пользователя образом. Использование технологий клиент-сервер позволяет сохранить значительные средства и время для получения необходимых данных, упрощают доступ и ведение.

Чтобы получить возможность использовать огромные объемы хранимой информации, кроме развития различных устройств, средств передачи данных, памяти, необходимы средства для обеспечения взаимодействия человека и машины, которые дадут возможность пользователю читать файлы, вводить запросы, добавлять новые данные, модифицировать хранимые данные или принимать решения на основании хранимых данных. Для обеспечения всех этих функций были созданы специализированные средства - системы управления базами данных (СУБД).

Целью курсового проекта является “Программная поддержка деятельности компании по предоставлению туристических услуг”.

# 1 ОПИСАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ

# Сущность и функция Туристической фирмы

Сперва стоит определиться с понятием информационная технология и информационная система. Информационная технология – это процесс, использующий совокупность средств и методов сбора, обработки и передачи данных для получения информации нового качества о состоянии объекта, процесса или явления. Цель информационной технологии – производство информации для ее анализа человеком и принятия на его основе решения по выполнению какого-либо действия.

Информационная система – среда, составляющими элементами которой являются компьютеры, компьютерные сети, программные продукты, базы данных, люди, различного рода технологические и программные средства.

Таким образом, информационная технология является более емким понятием, чем информационная система. Реализация функций информационной системы невозможна без знаний ориентированной на нее информационной технологии. Информационная технология может существовать и вне сферы информационной системы.

Внедрение персонального компьютера в информационную сферу и применение телекоммуникационных средств связи определили новый этап развития информационной технологии. Новая информационная технология — это информационная технология с «дружественным» интерфейсом работы пользователя, использующая персональные компьютеры и телекоммуникационные средства.

В данном курсовом проекте необходимо разработать информационную систему для туристической фирмы. Главной задачей при разработке данной информационной системы является быстрота нахождения информации, просмотра туров, услуги и, описывается проектирование " Программная поддержка деятельности компании по предоставлению туристических услуг". ИС должна быть проста в использовании и предельно понятна для сотрудников. Клиент не может вносить изменения в структуру таблиц и схему данных.

" Программная поддержка деятельности компании по предоставлению туристических услуг" должна обеспечивать выполнение следующих основных функций:

Ведение справочников номенклатуры городов, услуг, туров, транспорта, стран, сотрудников, заказов, клиентов;

* Вывод информации о стоимости туров;
* Вывод информации о турах в определенные страны;
* Формирование списка сотрудников, клиентов, туров.

# 2 ПОСТАНОВКА ЗАДАЧИ И ОБЗОР МЕТОДОВ ЕЁ РЕШЕНИЯ

# 2.1 Постановка задачи и обзор методов её решения

Задача данного курсового проекта – это создание функционирующего приложения в архитектуре клиент-сервер с организацией взаимодействия с базой данных на объектно-ориентированном языке Java. Серверное приложение может быть реализовано в виде консольного приложения или GUI-приложения. Клиентское приложение: оконное приложение с использованием стандартных библиотек пользовательского интерфейса (SWING, AWT).

Для успешной реализации системы необходимо, в первую очередь, выделить основные задачи, которые будет решать система «Туристической фирмы», а также те задачи, которые необходимо выполнить для правильной работы системы.

Основными задачами являются следующие:

* 1. Разработка базы данных для MySQL 5.5+.
  2. Разработка и использование собственной иерархии классов, расширение базовых классов, предоставляемых JDK.
  3. Реализация не менее 2-х паттернов проектирования.
  4. Использовать сокрытие данных (инкапсуляция), перегрузку методов, переопределение методов, социализацию, абстрактные типы данных (интерфейсы, абстрактные классы), статические методы, обработку исключительных ситуаций.
  5. В разрабатываемом приложении обеспечить добавление, редактирование и удаление записей из базы данных, сохранение табличных результатов в файле (создание текстового отчета), предоставление пользователю аналитической информации (графики, диаграммы).
  6. Предусмотреть возможность сохранять информацию в любой момент на сервере и загружать ранее сохраненные данные.
  7. В консольном окне серверной части должна отображаться текущая статистика подключений и параметры запуска сервера.
  8. Предусмотреть механизм авторизации пользователей (роли Администратор и Пользователь).
  9. Система должна предоставлять удобный интерфейс для конечного пользователя.

Рассмотрим некоторые функции, реализующие бизнес-логику:

* Регистрация пользователей.

При регистрации пользователю предоставляется окно регистрации, где необходимо создать логин и пароль. Данные отправляются на сервер и сохраняются в БД.

* Авторизация.

Пользователь вводит свой логин и пароль. Если введённый логин есть в базе данных, то дальше проверяем, совпадает ли пароль, введённый пользователем с паролем в базе данных. Если логин в БД не найдена или введён не верный пароль, то пользователю будет выведено сообщение об ошибке. Иначе выведется сообщение об успешном входе и в зависимости от роли пользователя, у него будут определённые привилегии.

* Просмотр информации.

Клинт посылает сообщения серверу о получения данные о заработной плате, после чего на сервере формируется SQL-запрос и выводится информация о клиентах.

* Просмотр операций.

Формируется SQL-запрос, и выводится информация, о об операциях.

* Визуализация информации посредством графиков и круговых диаграмм.

С сервера заращивается статистика об операциях и клиентах. После получения необходимых данных строятся графики о клиентах и операциях.

* Добавления клиентов.

Необходимо заполнить соответствующую форму и нажать кнопку «Добавить», после чего данные занесутся в базу данных.

**2.1.1 Swing**

Swing - это легкий набор инструментов для виджетов с графическим интерфейсом Java, который включает в себя богатый набор виджетов. Он является частью Java Foundation Classes (JFC) и включает в себя несколько пакетов для разработки многофункциональных настольных приложений на Java. Swing включает в себя встроенные элементы управления, такие как деревья, кнопки изображений, панели с вкладками, ползунки, панели инструментов, средства выбора цвета, таблицы и текстовые области для отображения HTTP или расширенного текстового формата (RTF). Компоненты Swing полностью написаны на Java и поэтому не зависят от платформы.

**Достоинства:**

* часть JDK, не нужно ставить дополнительных библиотек;
* по Swing гораздо больше книжек и ответов на форумах. Все проблемы, особенно у начинающих, гуглу досконально известны;
* встроенный редактор форм почти во всех средах разработки;
* на базе свинга есть много расширений типа SwingX;
* поддержка различных стилей (Look and feel).

**Недостатки:**

* окно с множеством компонентов начинает подтормаживать;
* работа с менеджерами компоновки может стать настоящим кошмаром в сложных интерфейсах.

Поначалу библиотеки Swing в Java вообще не было. Вместо нее использовалась библиотека AWT, общая идея которой была относительно неплохой. Важнейшим отличием Swing от AWT является то, что компоненты Swing вообще не нуждаются в поддержке операционной системы и потому гораздо более стабильны и быстры. Такие компоненты в Java называются легковесными, и понимание основных принципов их работы во многом объяснит работу Swing.

### 

### **2.1.2 AWT**

AWT была первой попыткой Sun создать графический интерфейс для Java. Они пошли легким путем и просто сделали прослойку на Java, которая вызывает методы из библиотек, написанных на С. Библиотечные методы создают и используют графические компоненты операционной среды. С одной стороны, это хорошо, так как программа на Java похожа на остальные программы в рамках данной ОС. Но с другой стороны, нет никакой гарантии, что различия в размерах компонентов и шрифтах не испортят внешний вид программы при запуске ее на другой платформе. Кроме того, чтобы обеспечить мультиплатформенность, пришлось унифицировать интерфейсы вызовов компонентов, из-за чего их функциональность получилась немного урезанной. Да и набор компонентов получился довольно небольшой. К примеру, в AWT нет таблиц, а в кнопках не поддерживается отображение иконок.

Использованные ресурсы AWT старается освобождать автоматически. Это немного усложняет архитектуру и влияет на производительность. Освоить AWT довольно просто, но написать что-то сложное будет несколько затруднительно. Сейчас ее используют разве что для апплетов.

**Достоинства:**

* часть JDK;
* скорость работы;
* графические компоненты похожи на стандартные.

**Недостатки:**

* использование нативных компонентов налагает ограничения на использование их свойств. Некоторые компоненты могут вообще не работать на «неродных» платформах;
* некоторые свойства, такие как иконки и всплывающие подсказки, в AWT вообще отсутствуют;
* стандартных компонентов AWT очень немного, программисту приходится реализовывать много кастомных;
* программа выглядит по-разному на разных платформах (может быть кривоватой).

В настоящее время AWT используется крайне редко — в основном в старых проектах и апплетах. Oracle припрятал обучалки и всячески поощряет переход на Swing. Оно и понятно, прямой доступ к компонентам оси может стать серьезной дырой в безопасности.

Вслед за AWT Sun разработала набор графических компонентов под названием Swing. Компоненты Swing полностью написаны на Java. Для отрисовки используется 2D, что принесло с собой сразу несколько преимуществ. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности. Стало легко создавать новые компоненты, наследуясь от существующих и рисуя все, что душе угодно. Стала возможной поддержка различных стилей и скинов. Вместе с тем скорость работы первых версий Swing оставляла желать лучшего. Некорректно написанная программа и вовсе могла повесить винду намертво.

**2.2 Диаграмма вариантов использования UML**

Первым шагом при описании функциональности системы является моделирование требований к ней.

Целями анализа и моделирования требований являются:

* достижение соглашения между разработчиками, заказчиками и пользователями о том, что должна делать ПС;
* достижение лучшего понимания разработчиками поведения ПС;
* ограничение системной функциональности;
* создание базиса для планирования разработки проекта;
* определение пользовательского интерфейса.

Для достижения этих целей используются диаграммы вариантов использования UML (Use case diagrams). На диаграммах вариантов использования (ВИ) изображаются актеры и варианты использования, между которыми существуют отношения. Актером будем называть внешнюю по отношению к ПС сущность, которая может взаимодействовать с системой. Актерами могут быть как люди, так и внешние системы или устройства. Следует всегда помнить, что актер – это не конкретный человек или устройство, а роль (должностная обязанность), в которой он выступает по отношению к программной системе.

Нахождение актеров – один из первых шагов в определении использования любой системы (как реальной, так и программной). Каждый источник внешних событий, с которыми должна взаимодействовать система, представляется как актер. Актер должен иметь имя, которое должно отражать его роль.

В данном случае актером является сотрудники предприятии.

В свою очередь, вариант использования (use case) служит для описания сервисов, которые система

В разрабатываемой системе предполагается два актёра: администратор системы и клиент.

Система управления туристической фирмы состоит из того, чтобы предоставить сотрудникам следующих действий:

авторизация – пользователь вводит свой логин и пароль. Если введённый логин есть в базе данных, то дальше проверяем, совпадает ли пароль, введённый пользователем с паролем в базе данных. Если запись в БД не найдена или введён не верный пароль, то будет выведено;

* управления данными о турах– сотрудники может добавлять, удалять, редактировать и просмотреть информацию о турах и операциях
* управления данными об операциях – сотрудники может добавлять, удалять, и просматривать информацию о билетах.
* просмотр операции по каждому заказу тура – сотрудники предприятии имеет возможность просмотреть произведённых операции по каждому туру отдельно.
* просмотр списка имеющихся сотрудников – сотрудники имеет возможность просмотреть данные о сотрудниках и клиентов.
* отмена действия – в процессе редактирования данных пользователь может отменить действию редактирования.
* управления доступами – администратор может разрешить либо запретит доступ к аккаунтам.
* управления сотрудниками предприятии – администратор может добавлять, удалять, редактировать и просматривать информацию о сотрудниках предприятии

Диаграмма вариантов использования данного курсового проекта представлена на рисунках приложения В.

Итак, смоделировав диаграмму вариантов использования, мы получили наглядное представление о нашей системе. Были определены действующие лица модели, и именно сотрудник и клиент.

**2.3 Модели представления системы и их описание**

### **2.3.1 Диаграмма классов**

Диаграмма классов является центральным звеном объектно-ориентированного подхода. Она является одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграммы классов используются при моделировании ПС наиболее часто. Они являются одной из форм статического описания системы с точки зрения ее проектирования, показывая ее структуру. Диаграмма классов не отображает динамическое поведение объектов изображенных на ней классов. На диаграммах классов показываются классы, интерфейсы и отношения между ними.

**2.3.2 Диаграмма последовательности**

Диаграммы последовательности визуально моделируют поток логики в системе, позволяя документировать и проверять свою логику, и обычно используются как для анализа, так и для целей проектирования. Диаграммы последовательности являются наиболее популярным артефактом UML для динамического моделирования, которое фокусируется на определении поведения в данной системе. Другие методы динамического моделирования включают в себя диаграмму активности, диаграмму связи, временную диаграмму и диаграмму обзора взаимодействия. Диаграммы последовательности, а также диаграммы классов и физические модели данных, на мой взгляд, являются наиболее важными моделями уровня проектирования для разработки современных Тур-приложений.

### **2.3.3 Диаграмма состояния**

Объекты меняют своё состояние в ответ на происходящие события и с течением времени. Диаграмма состояний представляет состояния объекта и переходы между ними, а также начальное и конечное состояние объекта.

Основными элементами диаграммы состояний являются «Состояние» и «Переход». Диаграмма состояний имеет схожую семантику с диаграммой деятельности, только деятельность здесь заменена состоянием, переходы символизируют действия. Таким образом, если для диаграммы деятельности отличие между понятиями «Деятельность» и «Действие» заключается в возможности дальнейшей декомпозиции, то на диаграмме состояний деятельность символизирует состояние, в котором объект находится продолжительное количество времени, в то время как действие моментально.

Цель данной диаграммы является показать поведение одного объекта в течение его жизни, начиная от создания объекта и заканчивая его уничтожением.

В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта.

## 2.4 Применение паттернов проектирования

Паттерн проектирования – повторимаяархитектурная конструкция, представляющая собой решение проблемыпроектированияв рамках некоторого часто возникающего контекста, иными словами, паттерны проектирования – это шаблоны необходимых шагов для решения каких-либо определенных задач.

При разработке системы туристической фирмы использовались следующие паттерны проектирования: Builder.

### **2.4.1 Паттерн MVC**

Паттерн MVC (Model/View/Controller) состоит из объектов трех видов. Модель – это объект приложения, а вид – экранное представление. Контроллер описывает, как интерфейс реагирует на управляющие воздействия пользователя.

Первым с клиентским запросом взаимодействует контроллер, контроллер разбивает данный запрос на элементы, далее он инициализирует объекты модели. После обработки данные отправляются на уровень представления.

Паттерн MVC – простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интерфейса.

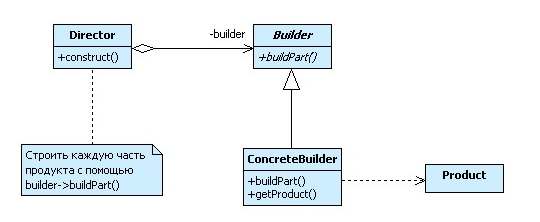
Диаграмма взаимодействия компонентов для паттерна «MVC» изображена на рисунке 2.3



Рисунок 2.3 – Диаграмма для паттерна «MVC»

**2.4.2 Паттерн Builder**

Builder определяет алгоритм поэтапного создания продукта в специальном классе Director (распорядитель), а ответственность за координацию процесса сборки отдельных частей продукта возлагает на иерархию классов Builder. В этой иерархии базовый класс Builder объявляет интерфейсы для построения отдельных частей продукта, а соответствующие подклассы ConcreteBuilder их реализуют подходящим образом, например, создают или получают нужные ресурсы, сохраняют промежуточные результаты, контролируют результаты выполнения операций.

Рисунок 2.4 - UML-диаграмма классов паттерна Builder.

**2.4.3 Паттерн Singleton**

Singleton относится к порождающим паттернам. Его дословный перевод – одиночка. Этот паттерн гарантирует, что у класса есть только один объект (один экземпляр класса) и к этому объекту предоставляется глобальная точка доступа. Из описания должно быть понятно, что этот паттерн должен применяться в двух случаях:

1. Когда в вашей программе должно быть создано не более одного объекта какого-либо класса. Например, в компьютерной игре у вас есть класс «Персонаж», и у этого класса должен быть только один объект описывающий самого персонажа.

2. Когда требуется предоставить глобальную точку доступа к объекту класса. Другими словами, нужно сделать так, чтобы объект вызывался из любого места программы. И, увы, для этого не достаточно просто создать глобальную переменную, ведь она не защищена от записи и кто угодно может изменить значение этой переменной и глобальная точка доступа к объекту будет потеряна. Это свойства Singleton'a нужно, например, когда у вас есть объект класса, который работает с базой данных, и вам нужно чтобы к базе данных был доступ из разных частей программы. А Singleton будет гарантировать, что никакой другой код не заменил созданный ранее экземпляр класса.

**2.4.4 Паттерн Фабричный метод**

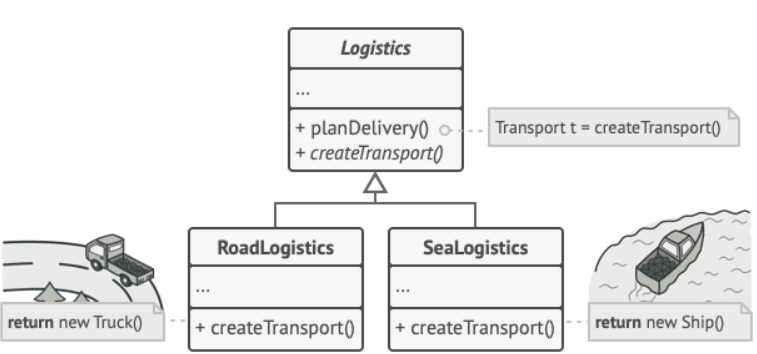
**Фабричный метод** — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Рисунок 2.5 - UML-диаграмма классов паттерна **Фабричный метод.**

# 3 ОПИСАНИЕ ОСНОВНОГО ПРОЦЕССА ФУНКЦИОНИРОВАНИЯ РАБОТЫ СИСТЕМЫ УПРАВЛЕНИЯ НА ОСНОВЕ СТАНДАРТА IDEF0

Анализ предметной области является одним из важнейших этапов проектирования информационной системы. На данном этапе необходимо понять и описать бизнес-логику предметной области. Известно, что исправление ошибок, допущенных на данной стадии, обходится примерно в десять раз дороже, чем на следующих, откуда следует, что начальные стадии проекта являются одними из наиболее критических и требуют применения эффективных средства автоматизации.

Для графического описания модели системы предприятии был использован стандарт **IDEF0** (Integration Definition for Function Modeling), который подставляет собой методологию функционального моделирования, предназначенную для описания бизнес-процессов.

**IDEF0** — это методология графического описания систем и процессов деятельности организации как множества взаимосвязанных функций. Она позволяет исследовать функции организации, не связывая их с объектами, обеспечивающими их реализацию.

В стандарте **IDEF0** посредством входа показывают объекты — информационные и материальные потоки, которые преобразуются в бизнес- процессе. С помощью управления показываются объекты — материальные и информационные потоки, которые не преобразуются в процессе, по нужны для его выполнения. Используя механизмы IDEF0 можно отображать инструменты и ресурсы, с помощью которых бизнес-процесс реализуется (например, технические средства, люди, информационные системы и т.д.). Выход бизнес-процесса, описанного в стандарте IDEF0, полностью соответствует по смыслу выходу процесса, описанному с помощью DFD-схемы.

Детальное рассмотрение процесса системы управления туристической фирмы предприятии представлен в приложение А. Оно реализовано с помощью нотации IDEF0, которая широко применяется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции. Нотация IDEF0 является одной из самых популярных нотаций моделирования бизнес-процессов. На первом уровне декомпозиции используется «контекстная диаграмма». Стрелки на этой диаграмме отображают связи объекта моделирования с окружающей средой.

# 4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ УПРАВЛЕНИЯ ТУРИСТИЧЕСКОЙ ФИРМЫ

**4.1 Описание информационной модели**

MySQL - это система управления реляционными базами данных или СУБД. В реляционной базе данных данные хранятся в отдельных таблицах, благодаря чему достигается высокая скорость и гибкость работы. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

СУБД MySQL использует традиционную архитектуру клиент/сервер, поэтому, работая с СУБД MySQL, пользователь реально работает с двумя программами. - Программой сервера базы данных, расположенной на компьютере, где хранится база данных. Она «прослушивает» запросы клиентов, поступающие по сети, и осуществляет доступ к содержимому базы данных для предоставления информации, которую запрашивают клиенты. - Клиентской программой, которая является программой, осуществляющей подключение к серверу и передающей запросы на сервер.

В реляционной модели база данных представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. В строках таблиц часть полей содержит данные, относящиеся непосредственно к записи, а часть — ссылки на записи других таблиц. Таким образом, связи между записями являются неотъемлемым свойством реляционной модели.

В реляционной модели достигается информационная и структурная независимость. Записи не связаны между собой настолько, чтобы изменение одной из них затронуло остальные, а изменение структуры базы данных не обязательно приводит к перекомпиляции работающих с ней приложений.

В реляционных СУБД применяется язык SQL, позволяющий формулировать произвольные, нерегламентированные запросы. Это язык четвертого поколения, поэтому любой пользователь может быстро научиться составлять запросы. К тому же, существует множество приложений, позволяющих строить логические схемы запросов в графическом виде. Все это происходит за счет ужесточения требований к производительности компьютеров.

Процесс разработки (проектирования) базы данных включает два этапа: разработку логической организации базы данных и создание ее на носителе. Логическая организация базы данных - это предоставление пользователя о предметной области, информация о которой должна храниться в базе данных.

Под физической организацией базы данных понимается совокупность средств и методов размещения данных во внешней памяти и на их основе внутренняя модель данных. Внутренняя модель является средством отображения логической модели данных, показывает, каким образом записи размещаются в базе данных, как они упорядочиваются, как организуются связи, каким путем можно осуществить выборку и так далее.

Проектирование базы данных системы управления Туристической фирмы начиналось с создания всех нужных таблиц в базе, всех полей, входящих в каждую таблицу, взаимодействия таблиц между собой с помощью специальных отношений и создание в соответствии с этими параметрами первичных и вторичных ключей.

**5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.**

**5.1 Руководство серверной части приложения**

Система устанавливается на компьютер с установленной 64-разрядной операционной системой Windows 10. Для работы серверной части системы требуется установленный MySQL версии 8.0.18 или выше и Java 12.0.2 или выше. Для клиентской части – Java 12.0.2 или выше. Сервер и клиент должны быть подключены к сети.

Во-первых, необходимо создать базу данных. Для этого можно воспользоваться скриптом генерации базы данных. По окончанию исполнения скрипта должна успешно создаться готовая к работе база данных.

Далее необходимо запустить сервер проекта. Для этого пользователь должен использовать server.jar файл. Текущее состояние работы сервера отображается в консольном окне.

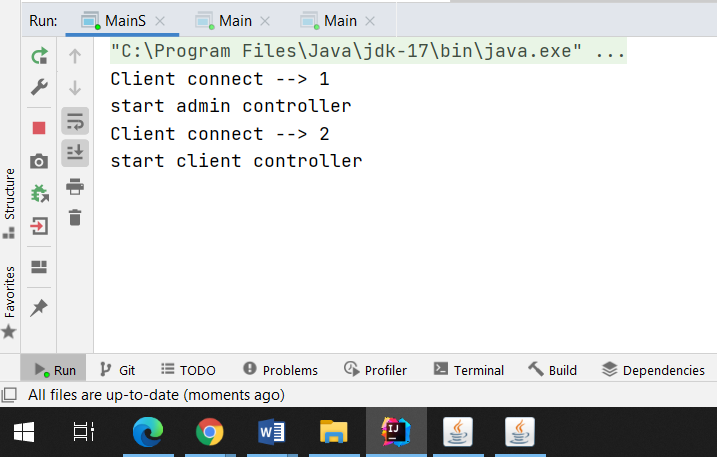


Рисунок 5.1 – Работа серверной части.

**5.2 Спецификация вариантов использования системы**

Диаграмма вариантов использования описывает взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующих в процессе.

В данном курсовом проекте были определены основные действующие лица: администратор и клиент.

Последнему предоставлена единственная функция регистрации в системе. Пользователь же уже знает, что он зарегистрирован и он может получить доступ к системе с правами либо администратора, либо клиента.

Администратор имеет функции, такие как:

- просмотр подробной информации о клиентах;

- работа с данными;

Администратору предоставлен более широкий функционал, для него реализованы возможности:

* добавление новых пользователей (клиентов или администраторов);
* добавление новых туров в систему;
* изменение данных пользователей, турах;
* удаление пользователей, билетов, туров.

# 6 Описание алгоритмов, реализующих бизнес-логику работы системы

**Алгоритм** — представляет собой пошаговую процедуру, которая определяет набор инструкций, которые должны быть выполнены в определенном порядке, чтобы получить желаемый результат. Алгоритмы, как правило, создаются независимо от базовых языков, то есть алгоритм может быть реализован на нескольких языках программирования.

**6.1 Алгоритм работы всей программы**

При запуске открывается главная окно, в котором пользователь должен пройти авторизацию. При успешной авторизации в зависимости от уровня доступа, пользователь получает разный набор возможностей для работы с программой. Если выполнен вход под администратором, то пользователь имеет доступ к базам данных тура, билетов и заказа, а также сотрудников включая логин и пароль и имеет возможность добавлять, удалять, редактировать данные в них. Если же была выполнена авторизация в качестве клиента, то функционал программы предоставляет возможность просматривать туры, билеты, заказы, а также отменять сделанные заказы.

**6.2 Алгоритм работы функции авторизации**

Для проведения авторизации пользователь указывает его логин и пароль, после ввода этих данных производится соединение с БД и на сервер отправляется запрос выборки данных о пользователях. Далее в цикле проверяется совпадение введенных данных с поступающими данными из базы данных. В случае несовпадения вызывается окно ошибки и пользователю предлагается ввести данные повторно.

**6.3 Руководство клиентской части приложения**

При запуске приложения появляется окно авторизации, в котором мы можем ввести данные аккаунта или зарегистрировать новый.

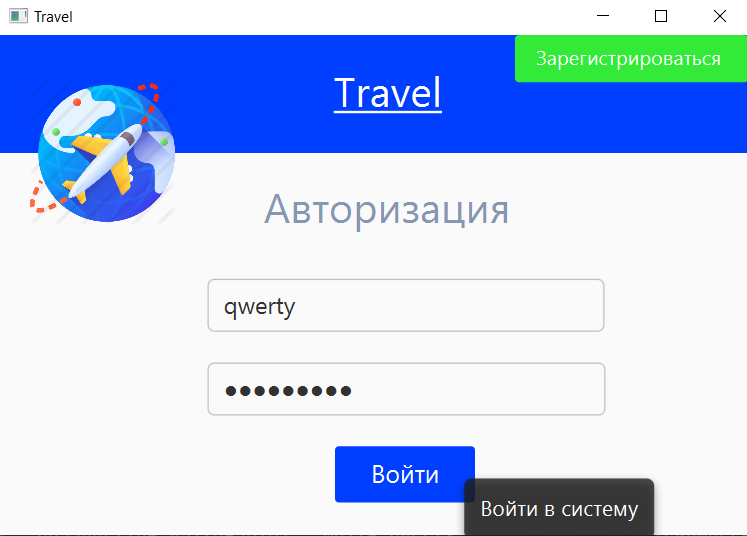


Рисунок 6.1 – Окно авторизации

Исходя из введённых данных серверная сторона, проверяет и определяет, доступ флага кто входит в систему администратор или клиент. Если входит клиент то, открывается главный страница меня, в котором определены все необходимые функции для пользования.

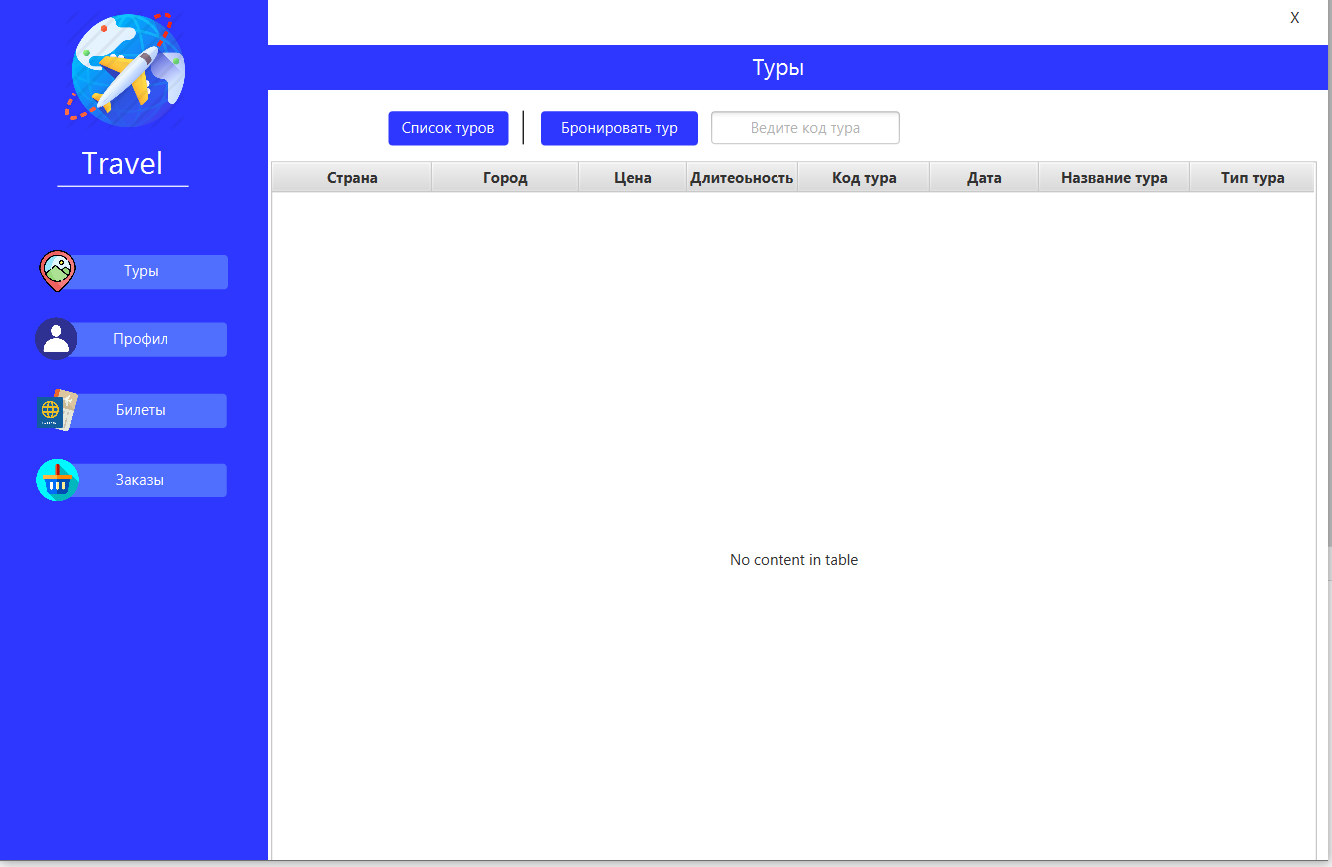


Рисунок 6.2 – Меня клиента

При нажатии на кнопе (Список Туров) том можно увидеть все туры.

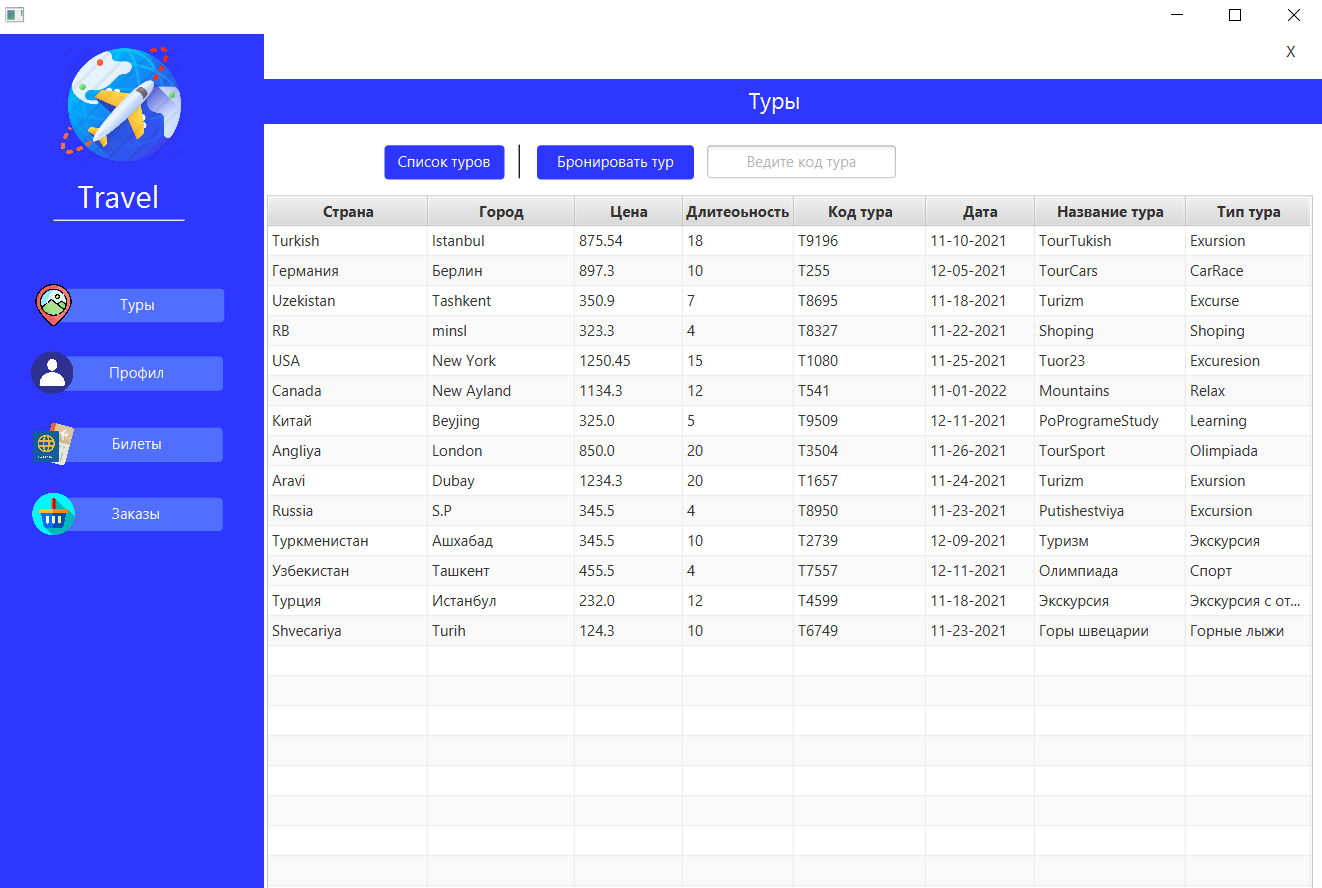


Рисунок 6.3 – Список актуальных туров

Можно также переходить между окнами меню. Чтоб забронировать тур сперва надо посмотреть какие виду туры есть и у каждого тура есть свой код тура, затем найдя для себя подходящий тур нужно ввести код тура в поля где написано (Ведите код тура) после ввода данных можно увидеть, что тур был забронирован.

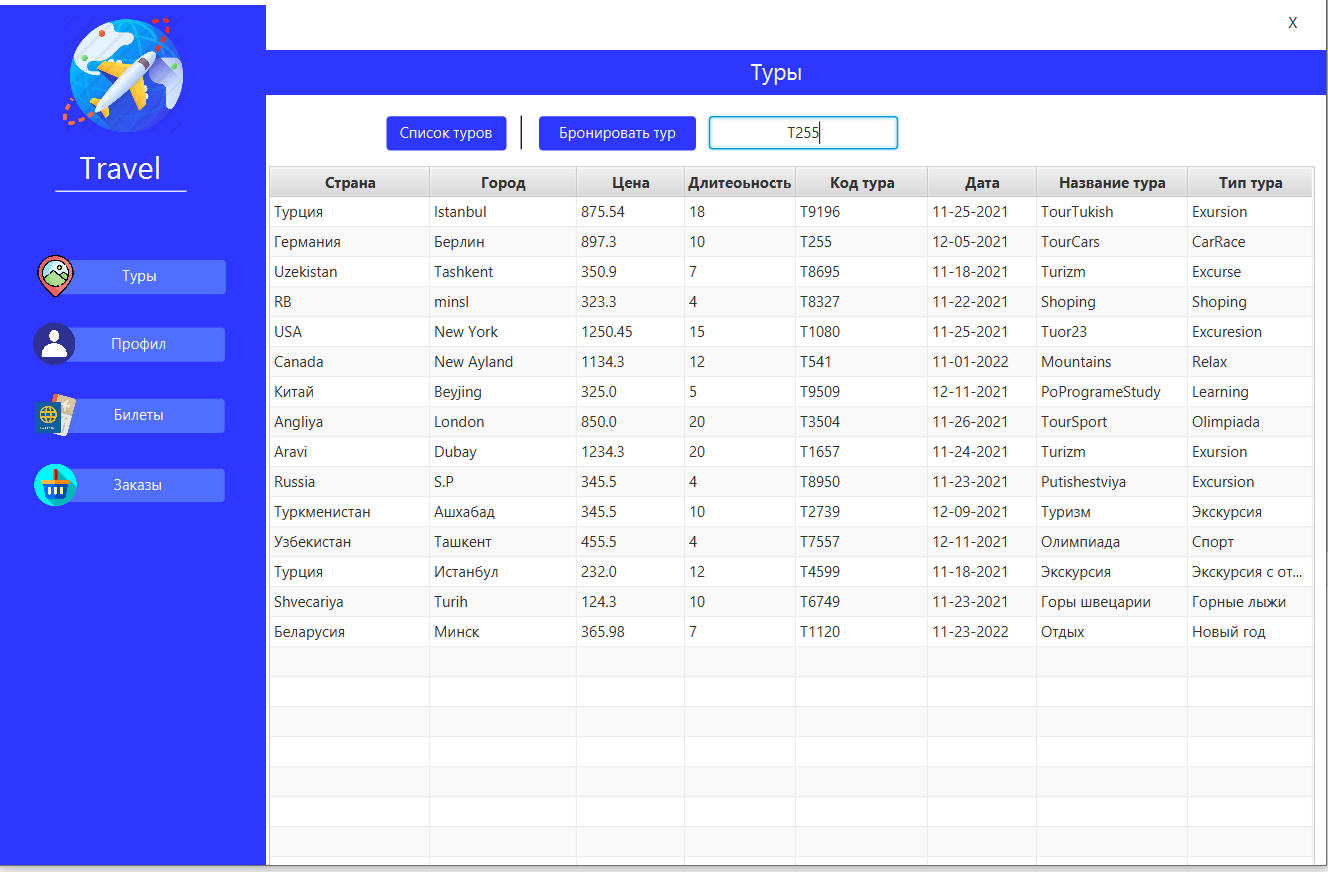


Рисунок 6.4 – Просмотр тура и выбор тура

Если пользователь ввел не правильные данные в полян (Ведите код тура), то отобразится ошибка заказа тура как показано в рисунке 6.5

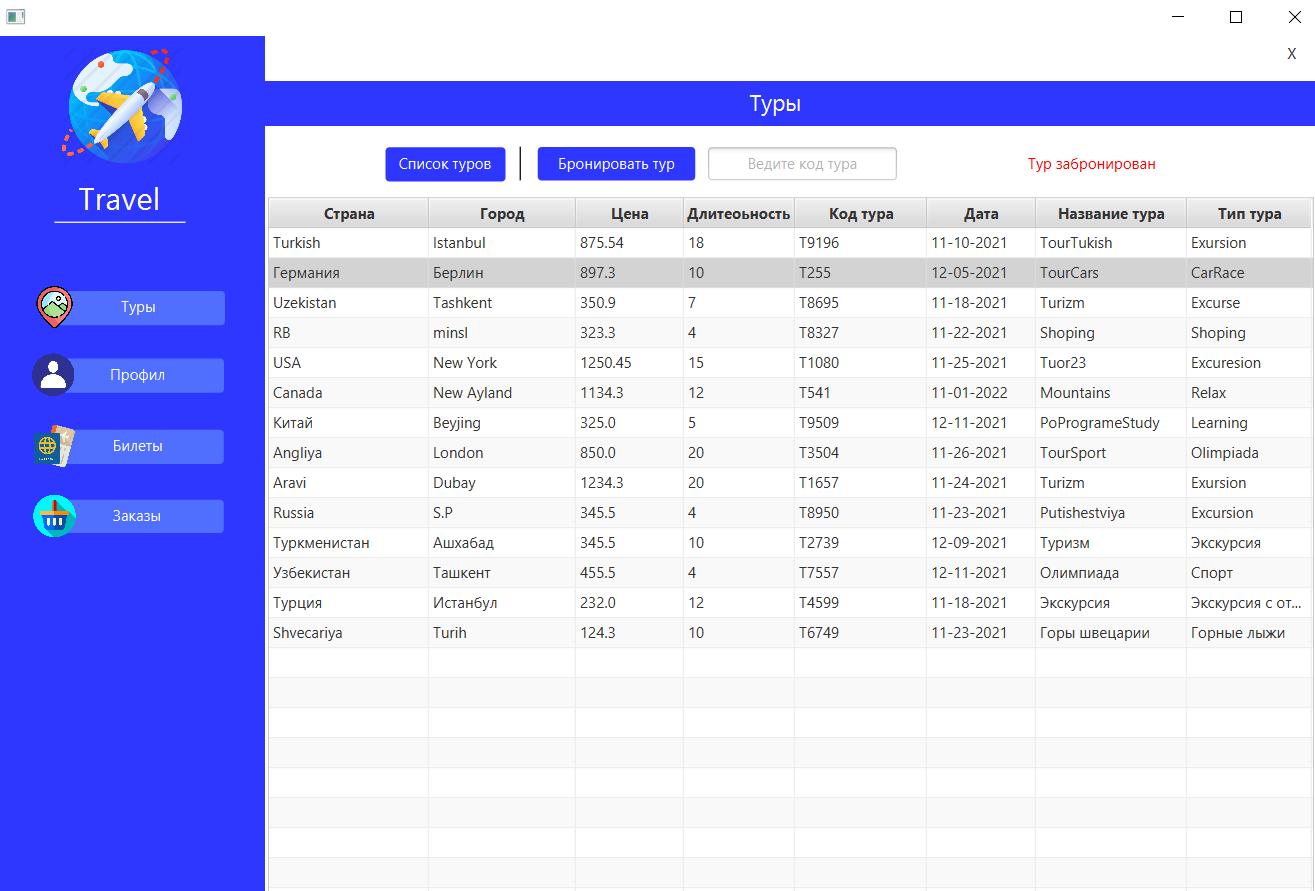
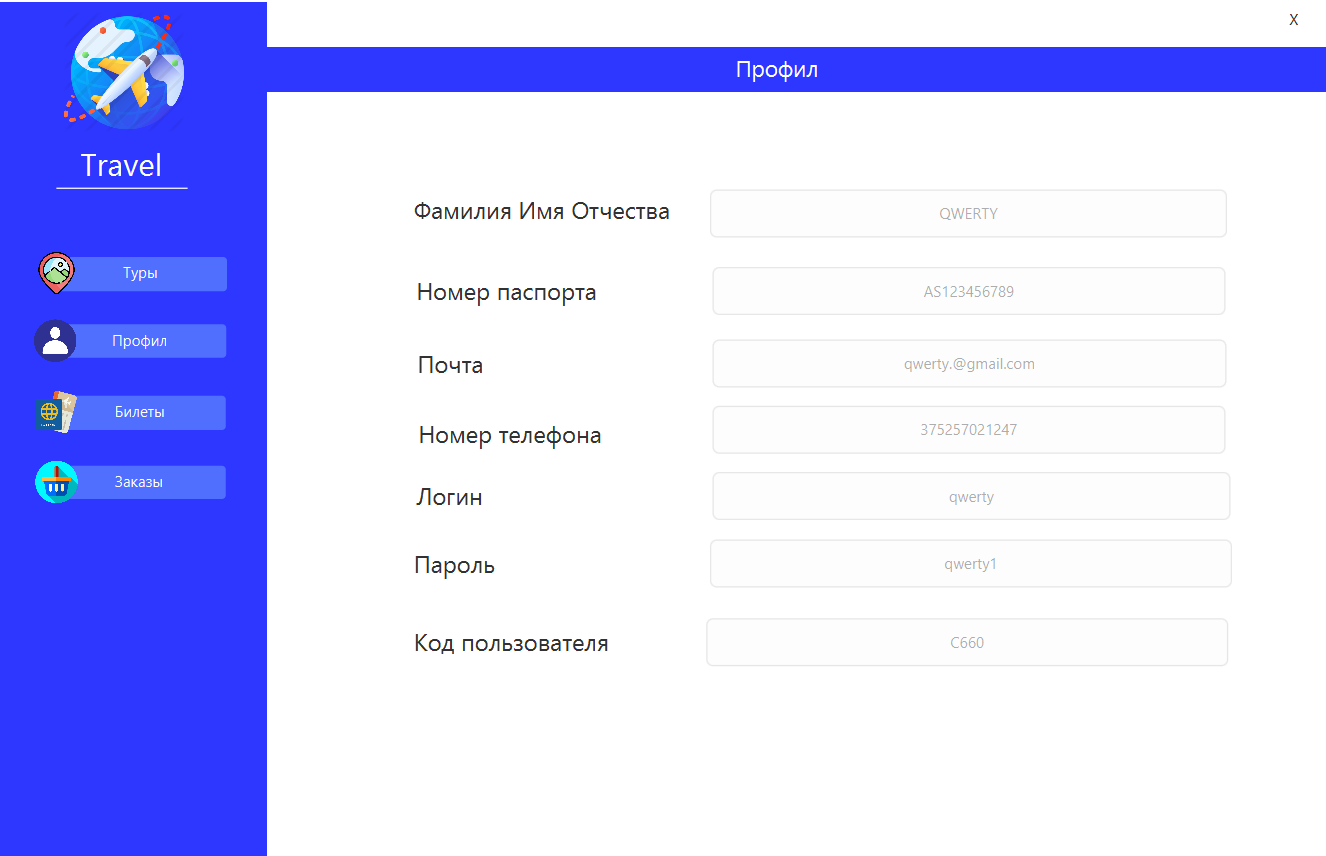


Рисунок 6.5 – Заказ тура

Также можно посмотреть свой личный кабинет пользователя как показано на рисунке 6.6. Клиент может посмотреть все свои билеты, которые он(она) заказал как показано на рисунке 6.7.

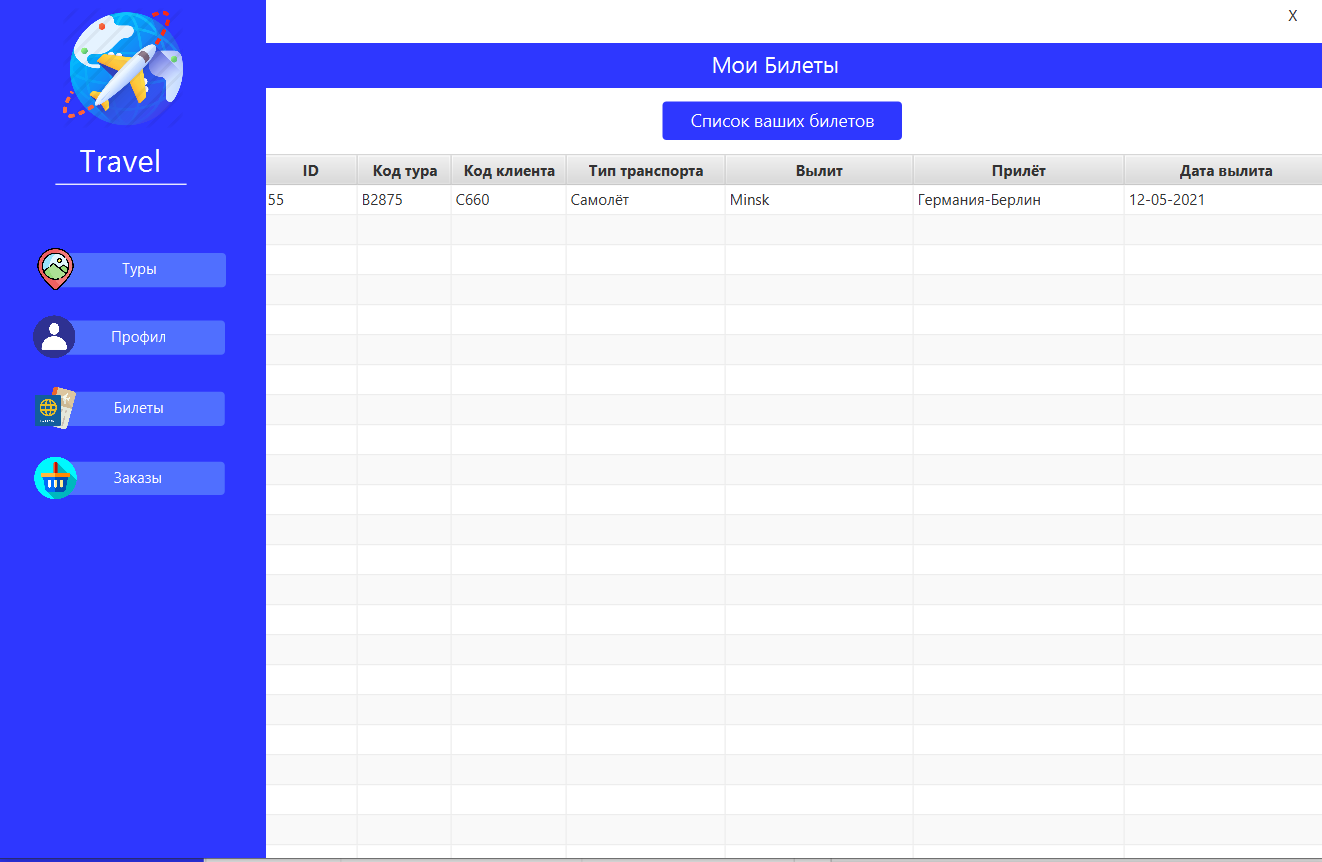
Рисунок 6.6 – Профиль пользователя

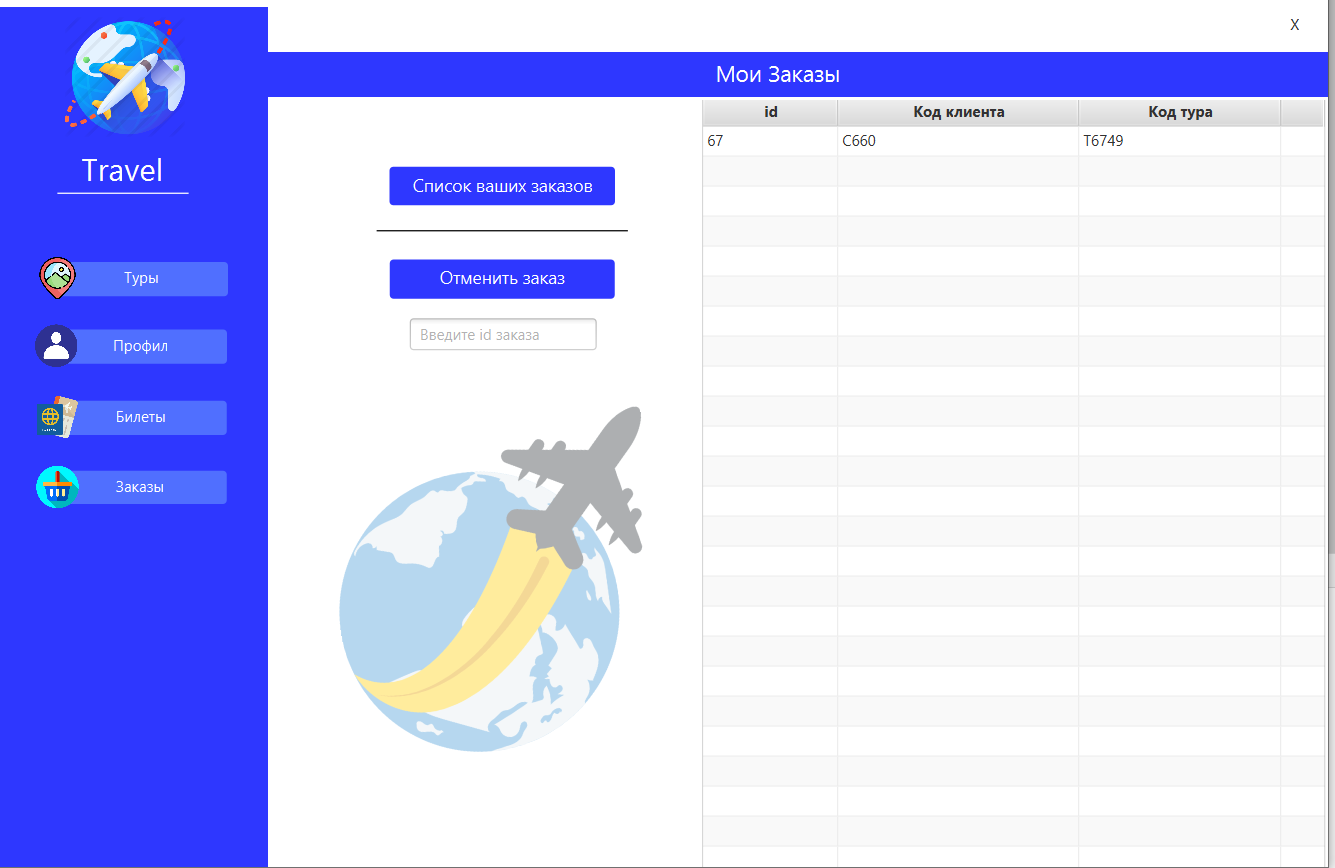
Рисунок 6.7 – Список билетов пользователя

И последний раздел меню Заказы, Пользователь может следить за своими заказами после того как он(она) забронировал(а) тур, нажав на кнопку (Список ваших заказов) как показано на рисунке 6.8.



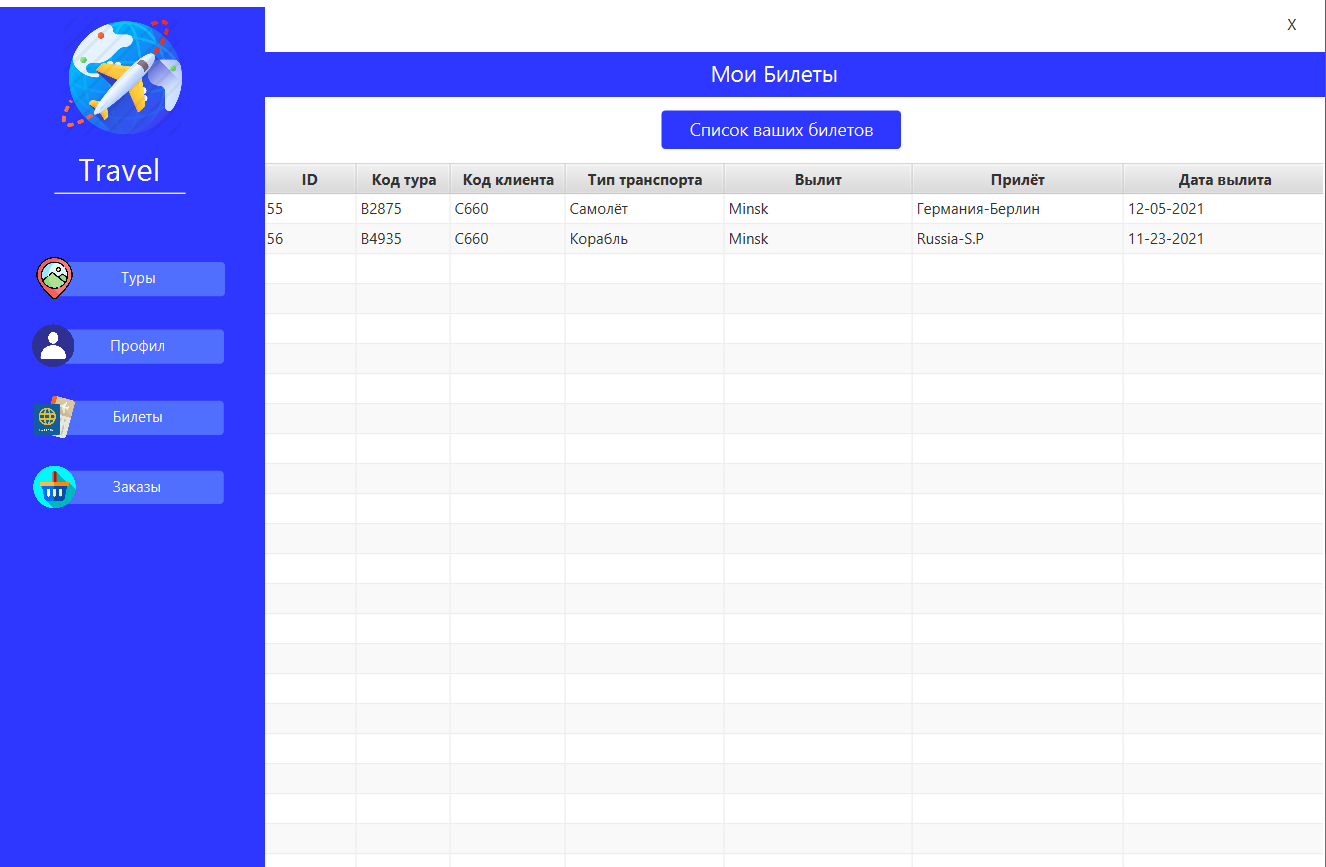
Рисунке 6.8. Список заказа пользователя

При просмотре заказа, заказ находится на стадии подтверждения администратором, при подтверждении заказа, администратор создаёт билет, а заказ удаляется с списка заказов как показано на рисунке 6.9.



Рисунке 6.9. Заказ подтверждён

Затем можно увидит свои билеты как показано на рисунке 6.10.



Рисунке 6.10 Билеты клиента

**6.4 Руководство администраторской части приложения**

При запуске приложения появляется окно авторизации, в котором мы можем ввести данные аккаунта или зарегистрировать новый.

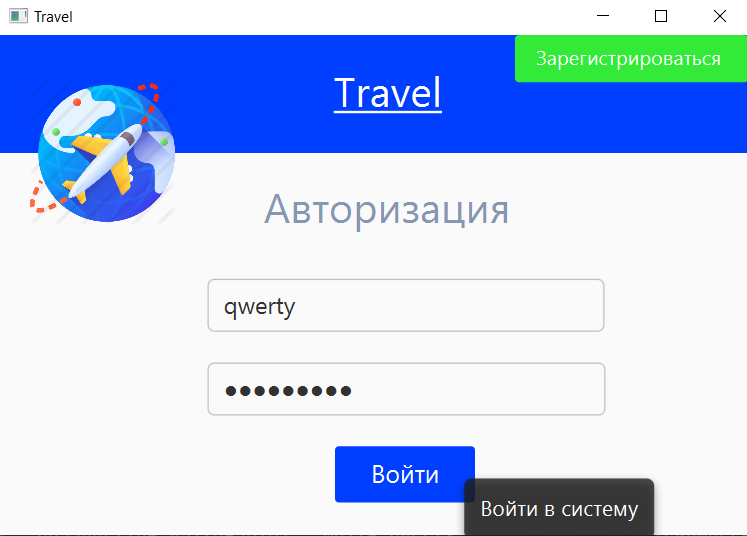


Рисунок 6.11 – Окно авторизации

Исходя из введённых данных серверная сторона, проверяет и определяет, доступ флага кто входит в систему администратор или клиент. Если входит администратор то, открывается главный страница меня, в котором определены все необходимые функции для пользования.

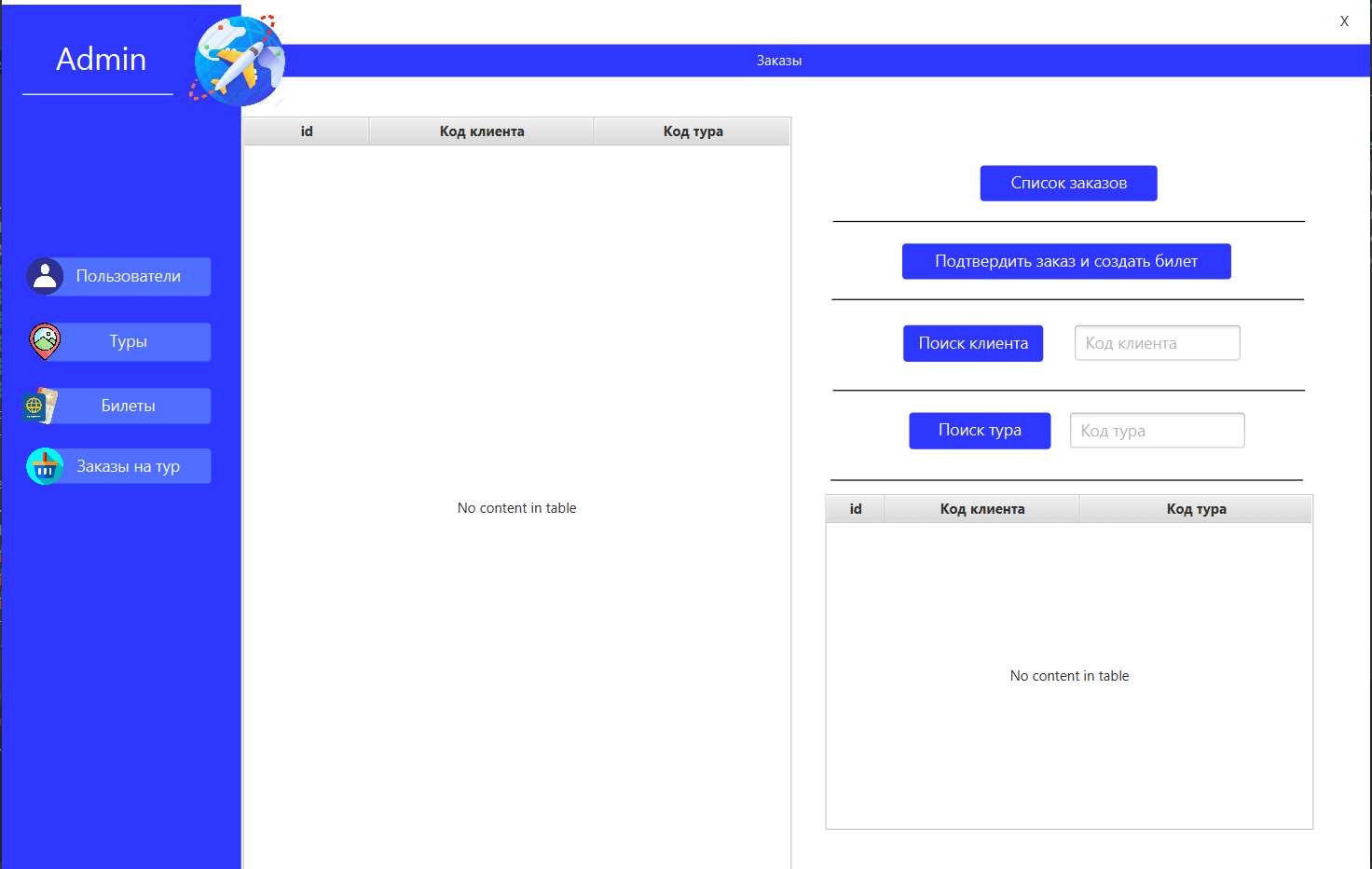


Рисунок 6.12 – Главный меню

При нажатии на кнопе (Пользователи, Туры, Билеты, Заказы на тур) том можно перемешаться между окнами исходя нажатой кнопки, например, кнопка(Пользователи), также можно увидеть раздел окна изменить и добавить.

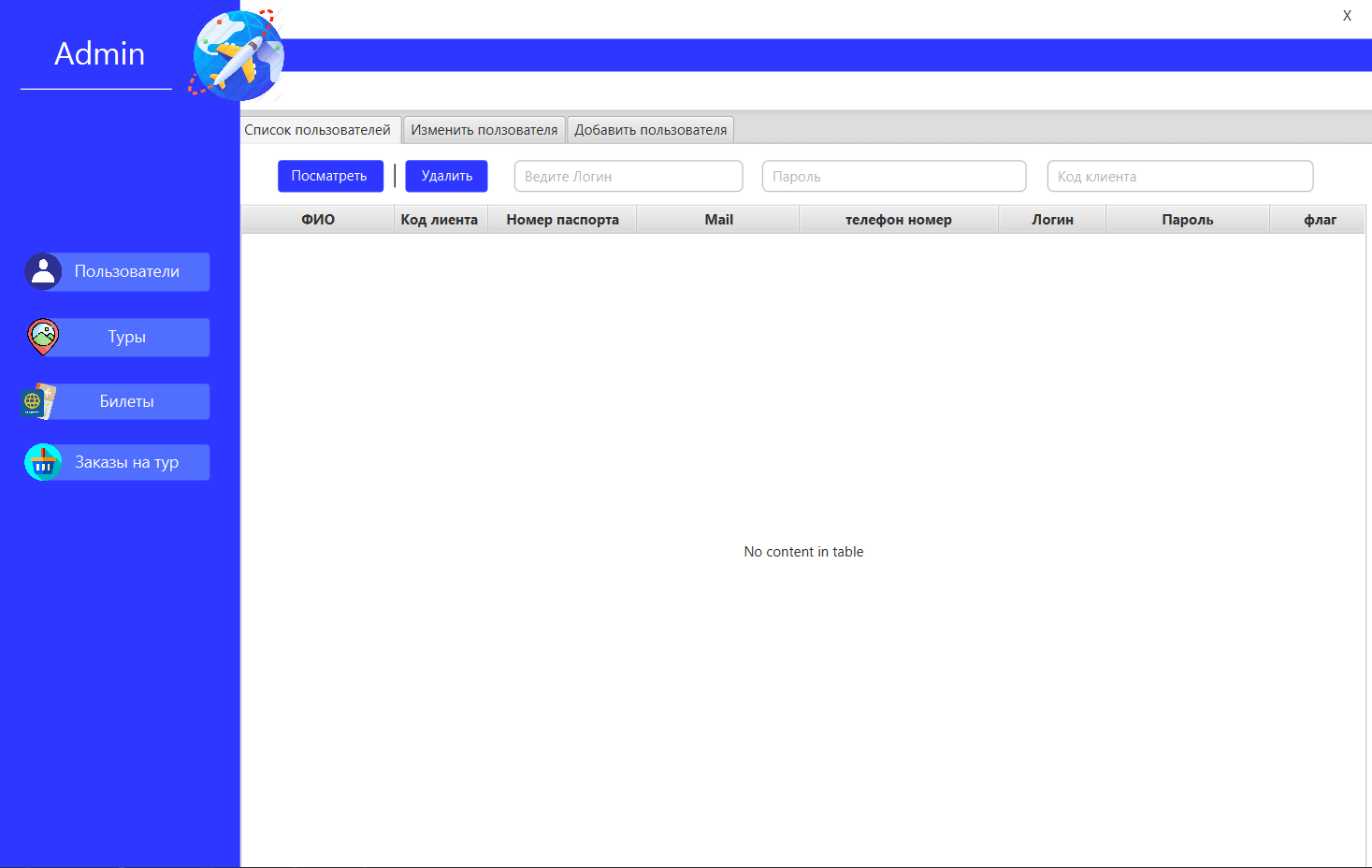


Рисунок 6.13 – Окно пользователей

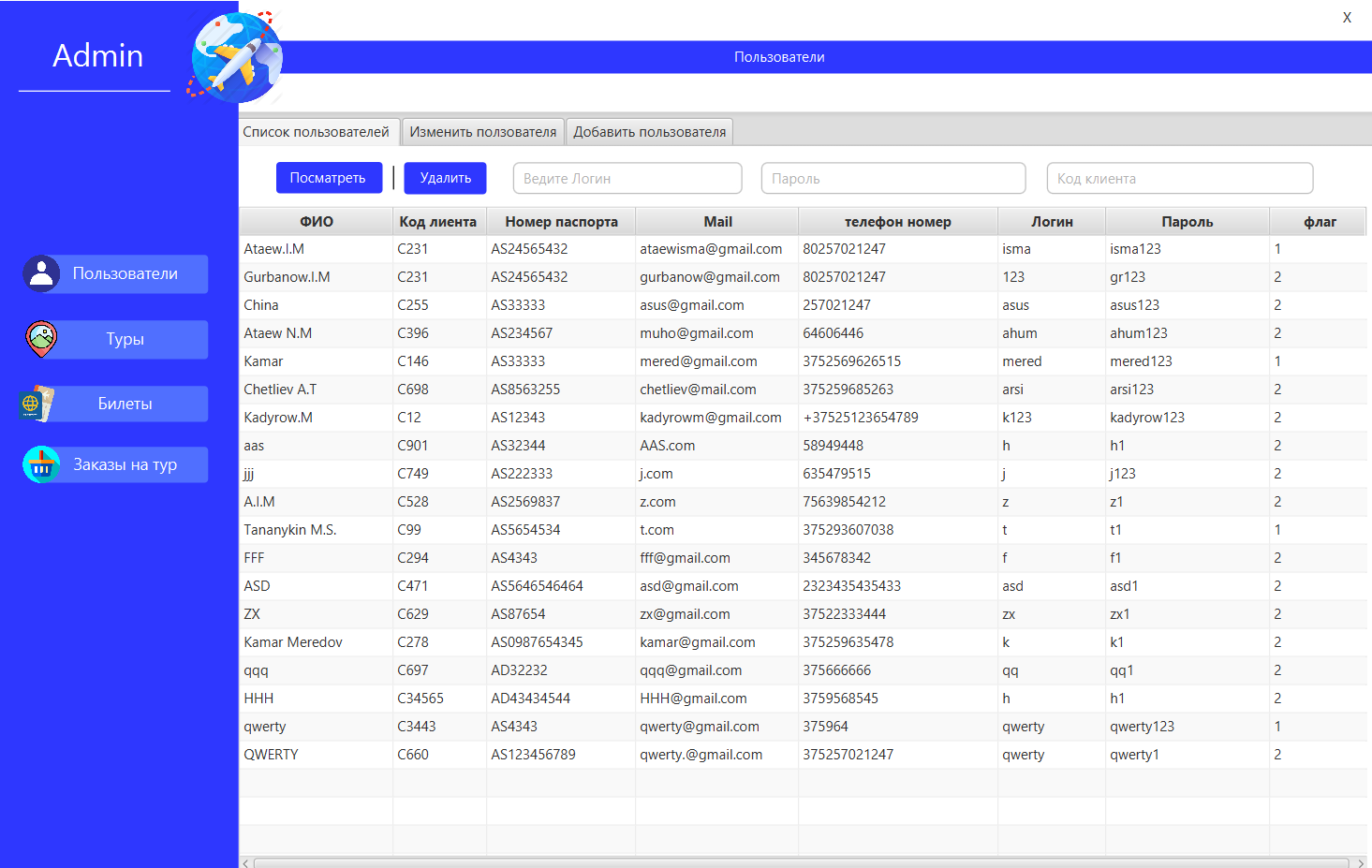
при, нажатия на кнопку (Посмотреть), то можно увидеть список всех пользователей как показано на рисунке.

Рисунок 6.14 – Список пользователей

Если при удалении пользователя вести не корректные данные, то на экран где ввели данные для удаления произведет анимацию влево и на право так можно понять, что ведены не корректные данные или оставить пустыми и нажать на кнопку удалить.

Так же администратор может добавлять нового пользователя как показано на Рисунок 6.15.

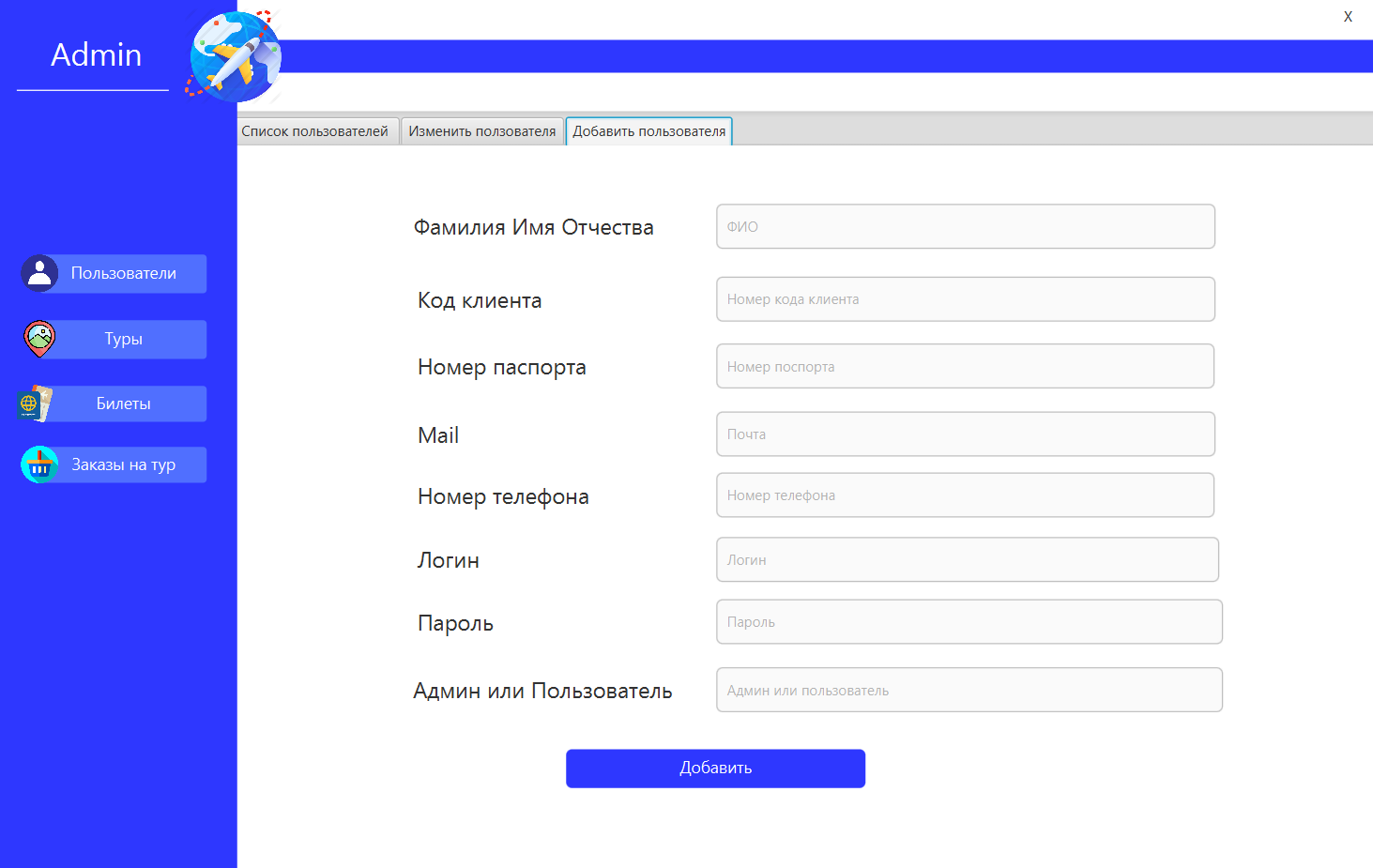


Рисунок 6.15 – Окно добавление пользователя

Если оставить поля пустыми или где надо вести цифры ведут символы, то система определит, что некорректные данные ведены и, выведет сообщение об ошибке на экран как показано на рисунке.

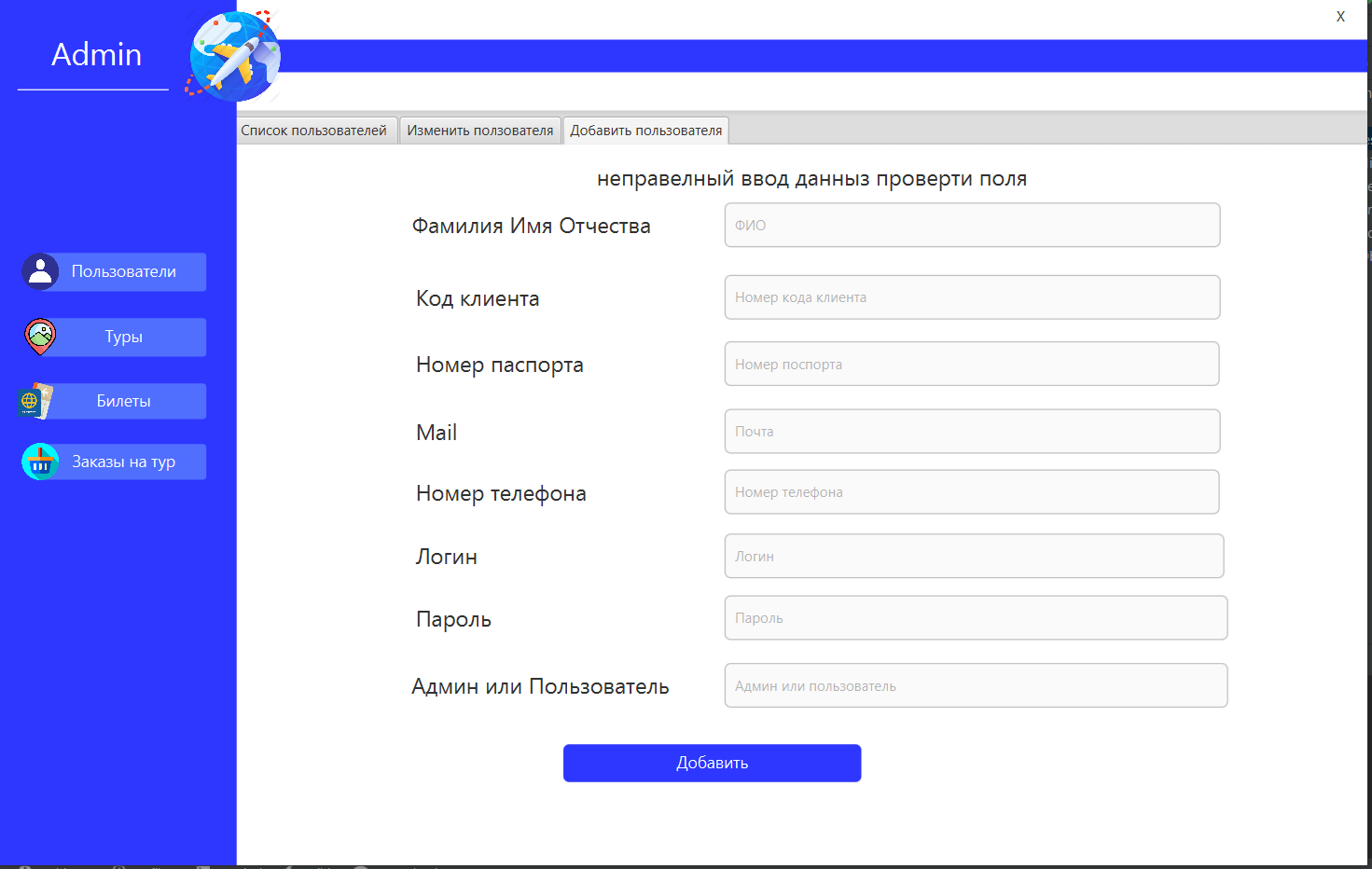
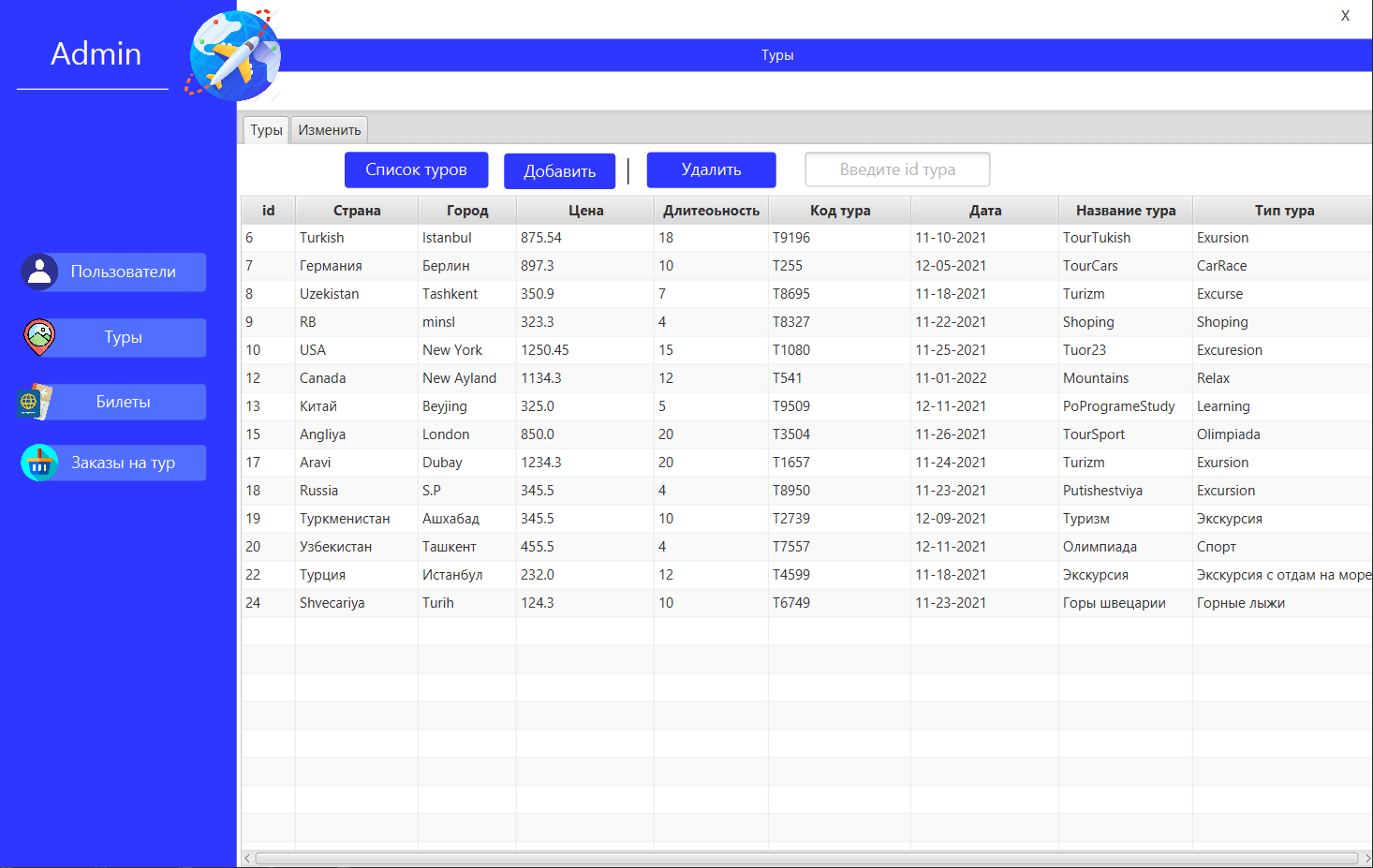


Рисунок 6.16 – Окно добавление пользователя



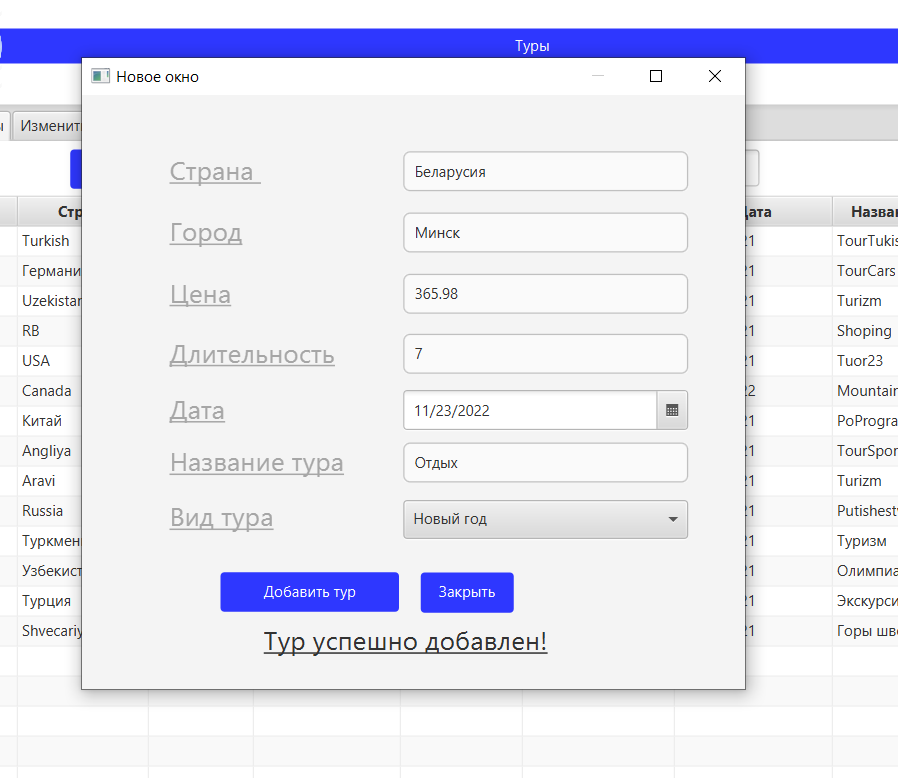
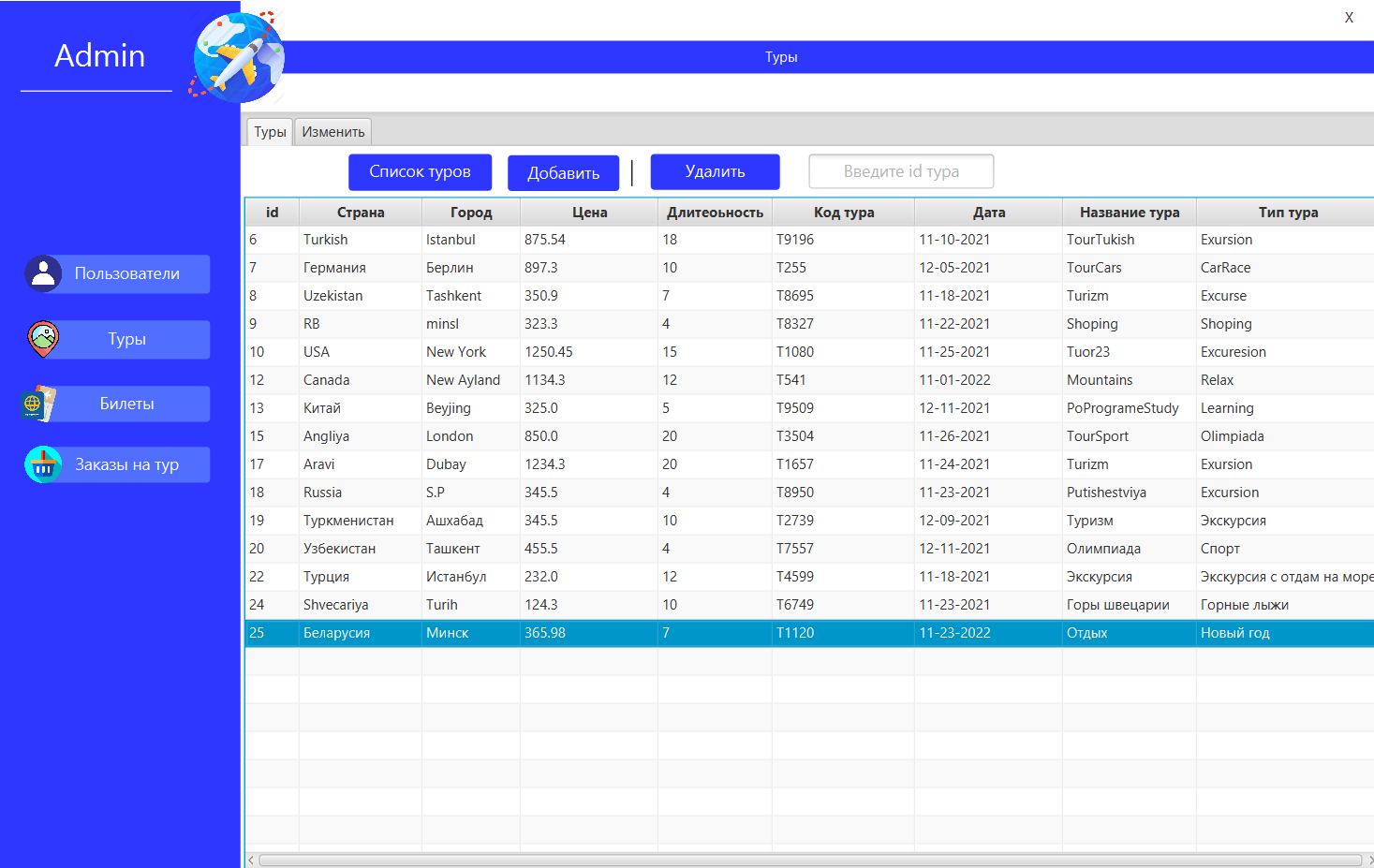
Рисунок 6.17 – Окно туров

Рисунок 6.18 – Окно добавление тура



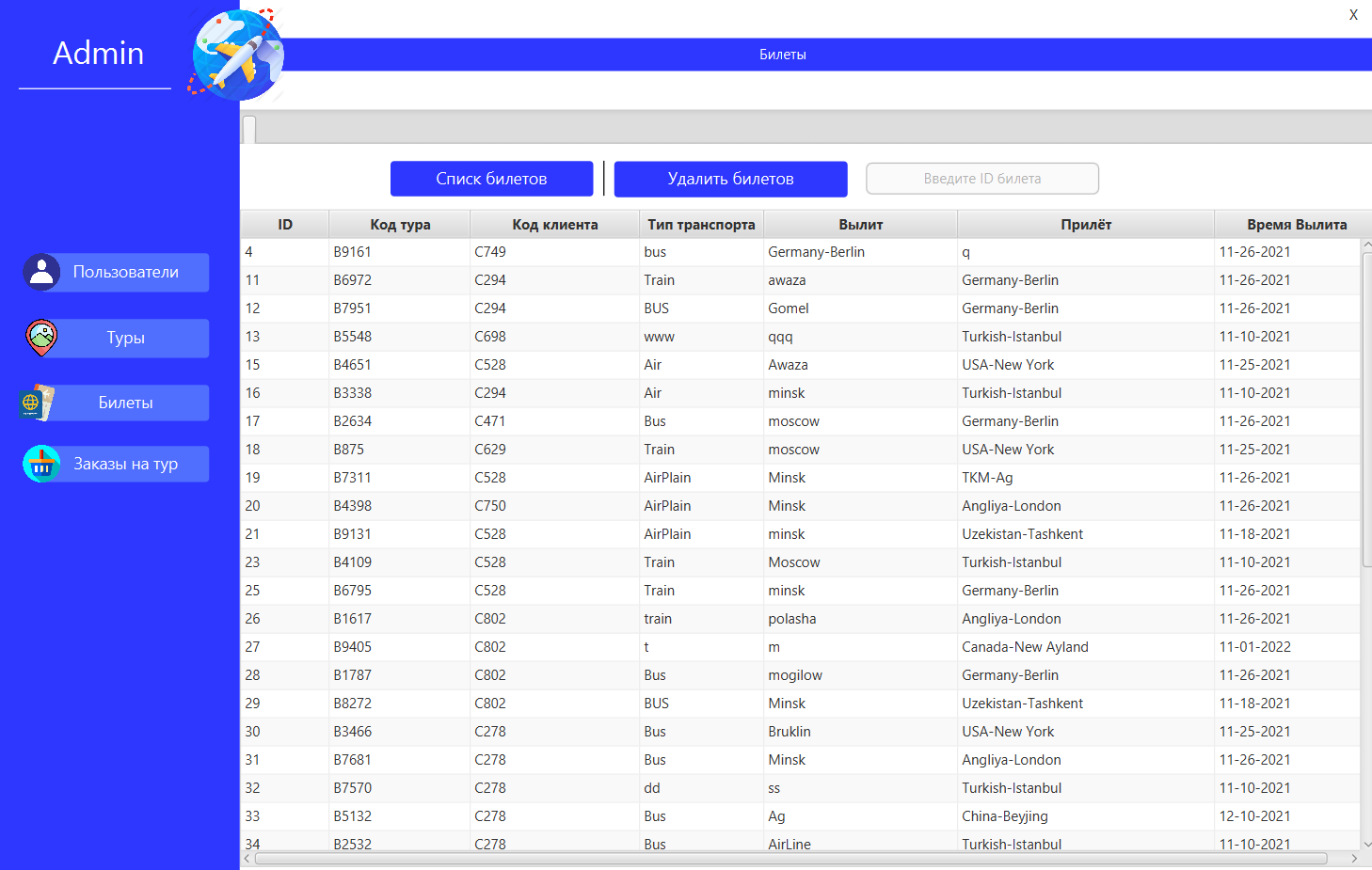
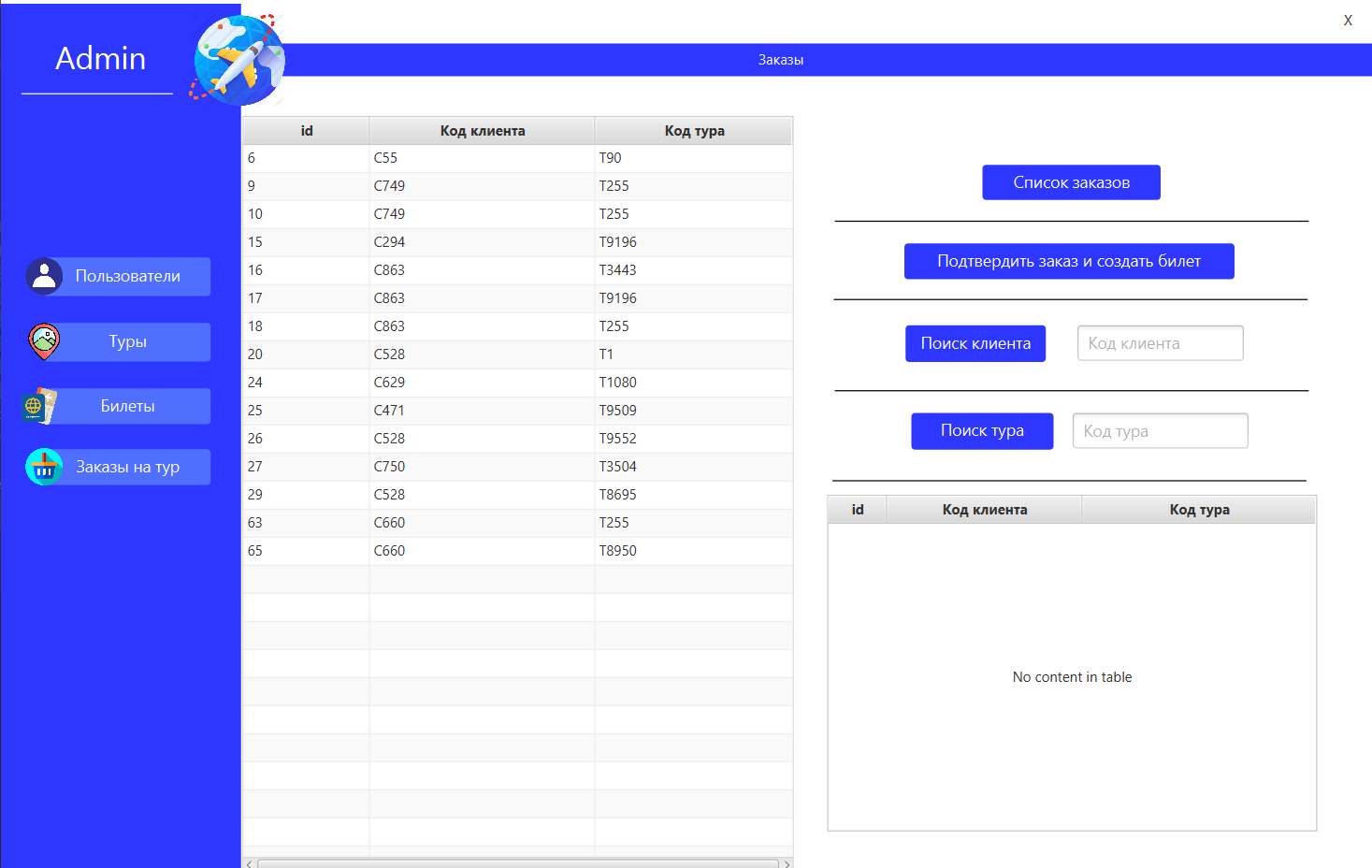
Рисунок 6.19 – Список туров

Рисунок 6.20 – Список билетов



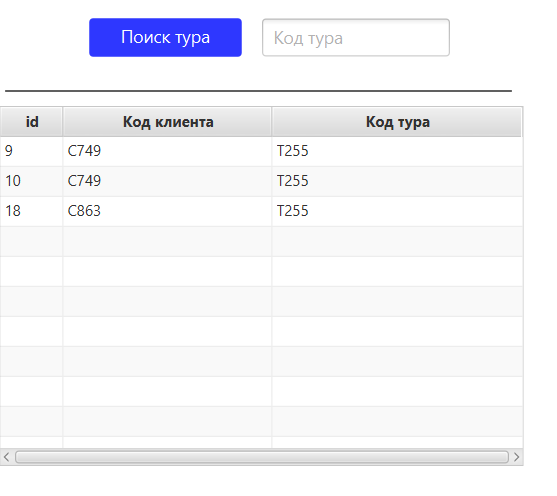
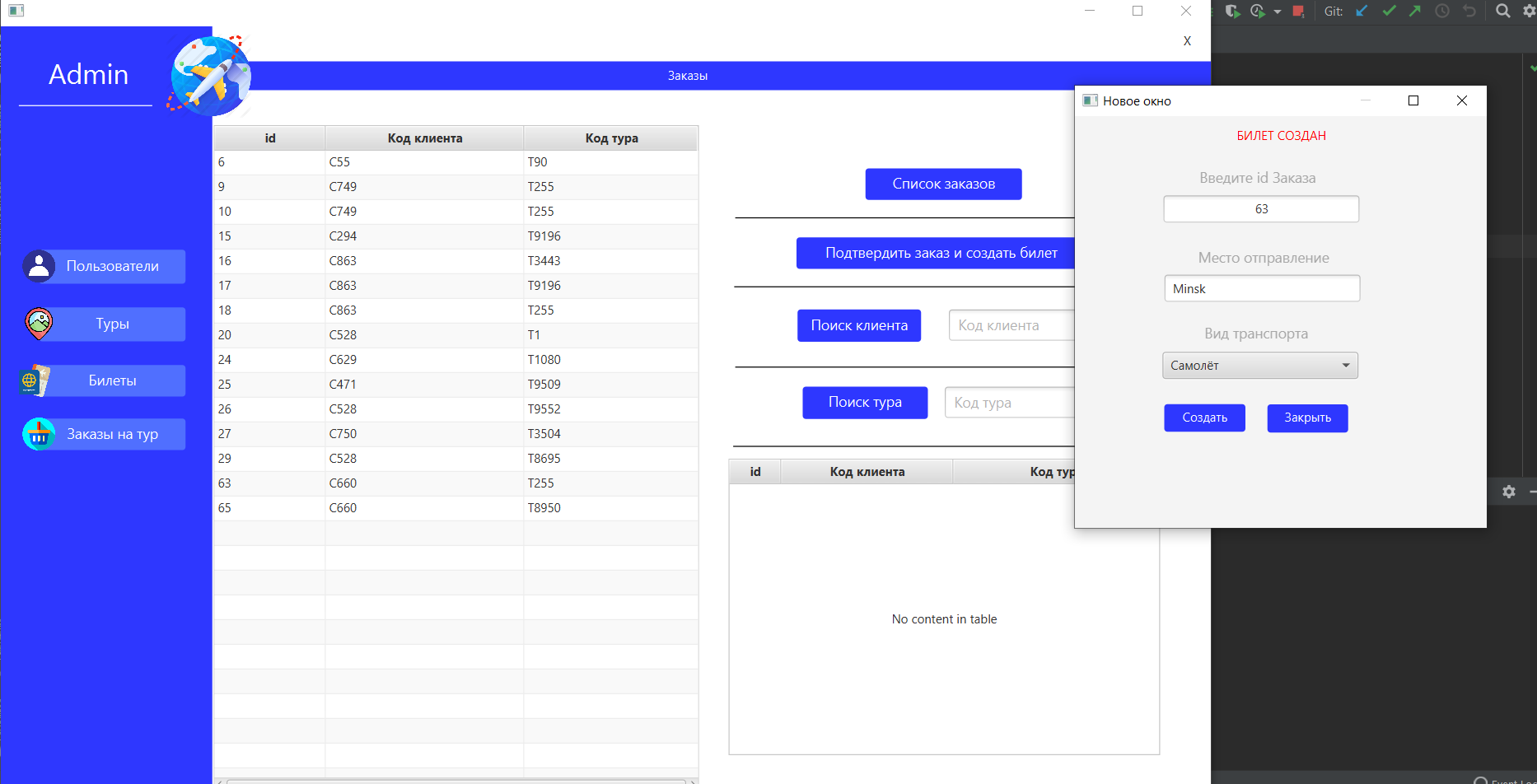
Рисунок 6.21 – Список заказов

Рисунок 6.22 – Список заказов



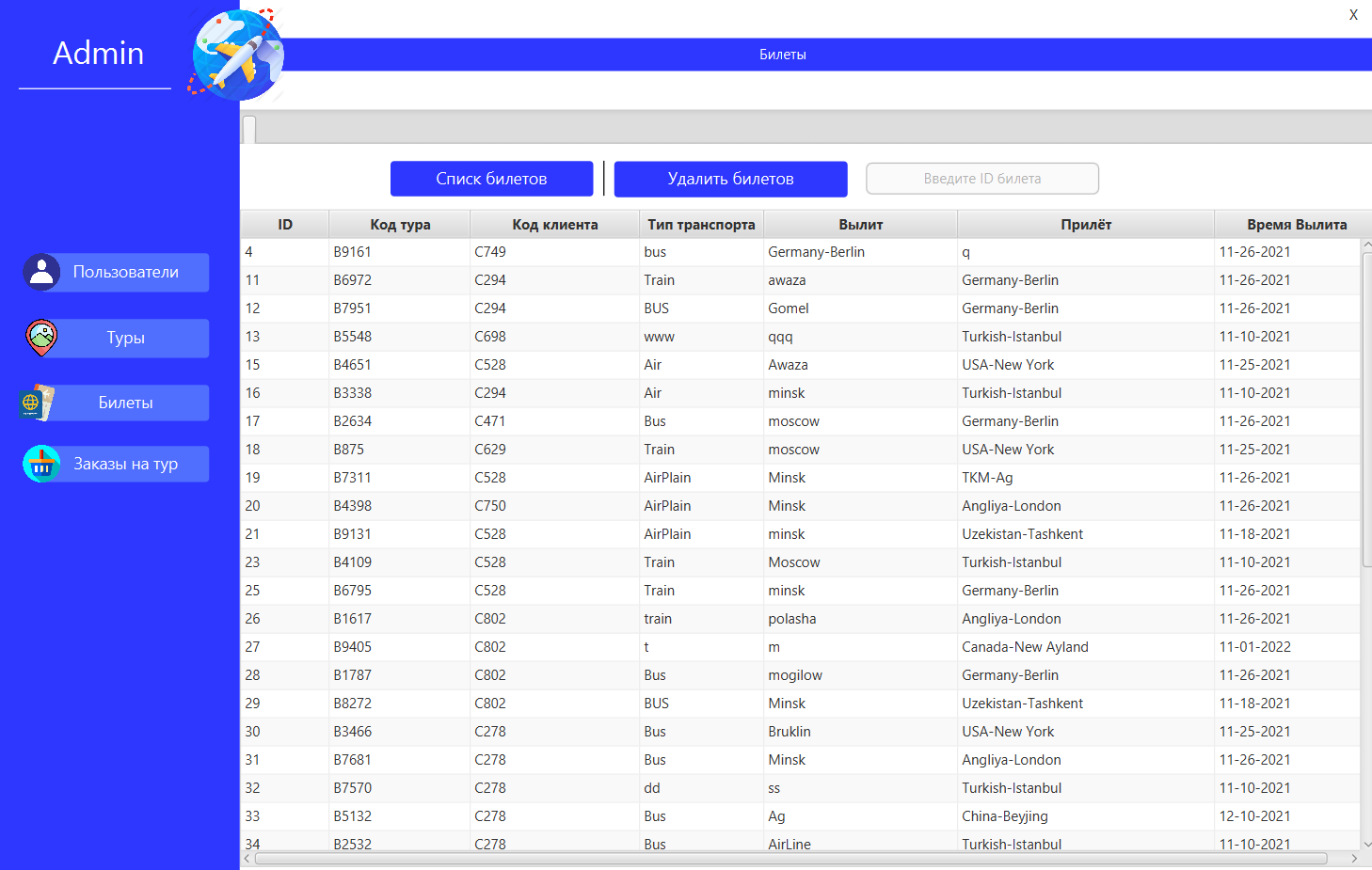
Рисунок 6.23 – Подтверждение заказа и создание билета

Рисунок 6.24 – Список билетов

**Заключение**

Прогресс, достигнутый за последние несколько лет во всех аспектах вычислительной техники, включая теорию, технологию и приложения, привели к значительному расширению области применения компьютеров и росту числа их пользователей. Существенной частью современного общества являются разнообразные системы доступа и хранения информации, которые являются неотъемлемой составляющей современного научно-технического прогресса. Существует много веских причин перевода существующей информации на компьютерную основу, так как более быстрая обработка данных и централизация их хранения с использованием клиент/серверных технологий позволяют сберечь значительные средства, а главное и время для получения необходимой информации, а также упрощает доступ и ведение.

В результате проделанной работы был проведен анализ предметной области, а именно системы управление тур фирмы.

При этом задачи, поставленные перед началом выполнения, были достигнуты следующим образом:

* в ходе программы реализован множественный функционал, в разной степени доступный пользователям в зависимости от их прав;
* программный продукт позволяет сотрудникам записать данные
* данный продукт может быть использован, как сотрудниками клиники, так и простыми пользователями;
* предусмотрена обработка ошибок, которые могут возникнуть в процессе работы;
* применение базы данных в качестве хранилища информации позволяет оптимально и эффективно хранить информацию, ее структурировать;
* безопасность обеспечивается за счет разграничения прав доступа, системы авторизации;
* для удобства пользователя, программа располагает достаточно понятным, наглядным и удобным интерфейсом взаимодействия пользователя с программой;
* интерфейс программы поддерживает русский язык;

Данное приложение позволит повысить производительность и качество работы туристической фирмы, поможет работникам тур фирмы в решении практических задач, позволит снизить трудозатраты, улучшит качество и повысит скорость обслуживания клиентов.

В процессе разработки, приложение было протестировано, в результате не было выявлено фатальных ошибок, а все выявленные незначительные ошибки были устранены.

Все поставленные задачи решены. В будущем приложение будет совершенствоваться посредством улучшения и расширения информационной модели системы, оптимизации имеющихся функций и добавления новых. Также будет улучшен интерфейс приложения.

Помимо бизнес-логики также были определены требования к работе, внешнему виду и содержанию системы, а также были определены основные сущности системы для организации моделей.

Подводя итог всему выше сказанному, можно утверждать, что разработка таких программ позволяет значительно оптимизировать и упростить деятельность подобных учреждений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://en.wikipedia.org/wiki/Java>

[2] Документация по архитектуре клиент-сервер [Электронный ресурс] – Режим доступа:   
http://www.4stud.info/networking/lecture5.html

[3] Комличенко, В.Н. Компьютерные сети:лаб. практикум, Минск: БГУИР, 2012.

[4] Живицкая, Е.Н., Комаровский А.О., Швед О.И. Системный анализ и проектирование информационных систем. Лабораторный практикум: учеб-метод. Пособие – Минск: БГУИР, 2011.

[5] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://str.by/all-news/streamline-22-years?gclid=CjwKCAiAnvj9BRA4EiwAuUMDfy\_WrzzkC\_an0PE9gBas3VxCA9NkA0cJbt2tAxy1\_LoLieUz9dXGZBoCqrkQAvD\_BwE

[6] [Электронный ресурс]. – Электронные данные. - Режим доступа: <http://www.ru.wikipedia.org>;

[7] [Электронный ресурс]. – Электронные данные. - Режим доступа: http://qt-doc.ru - “Модель "клиент-сервер";

[8] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://openjfx.io/

[9] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://www.mysql.com/products/workbench/

[10] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://valor-software.com/ngx-bootstrap/#/>

[11] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://fontawesome.com/>;

[12] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://getbootstrap.com/>;

[13] [Электронный ресурс]. – Электронные данные. - Режим доступа: <https://www.mysql.com/>;

[14] [Электронный ресурс]. – Электронные данные. - Режим доступа: https://www.instituteiba.by/courses/languages/

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Функциональная модель процессов функционирования системы управления туристической фирмы**

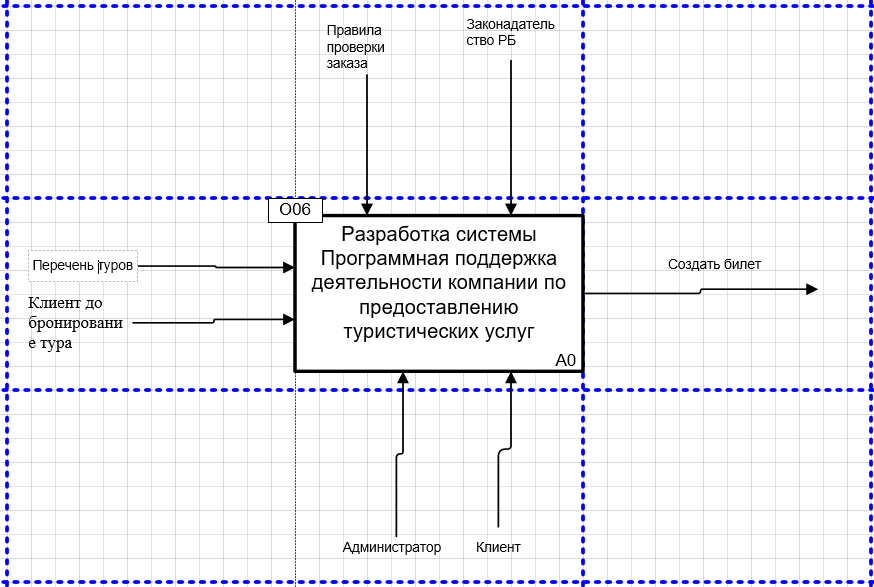


Рисунок А.1 - Контекстная диаграмма функциональной модели

Продолжение приложения А

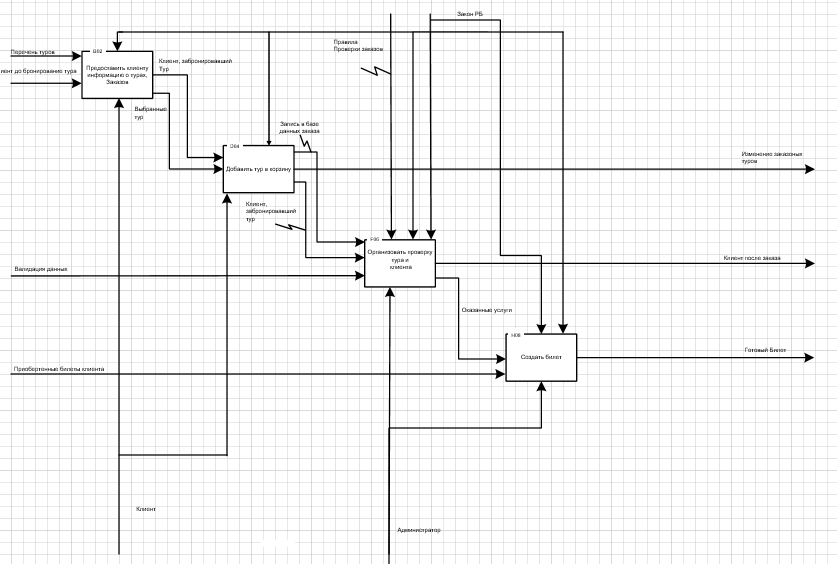


Рисунок А.2 – Декомпозиция контекстной диаграммы верхнего уровня

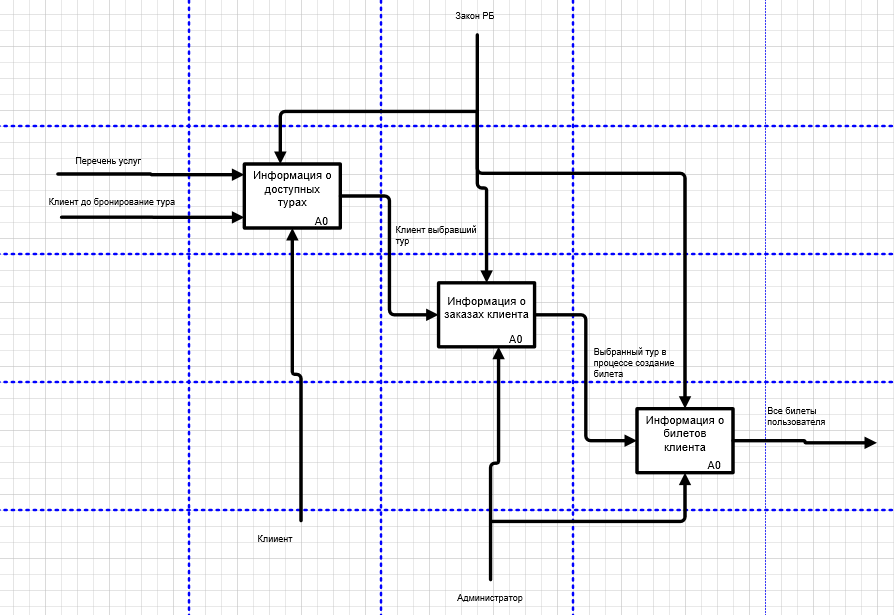


Рисунок А.3 – Декомпозиция процесса туристической фирмы

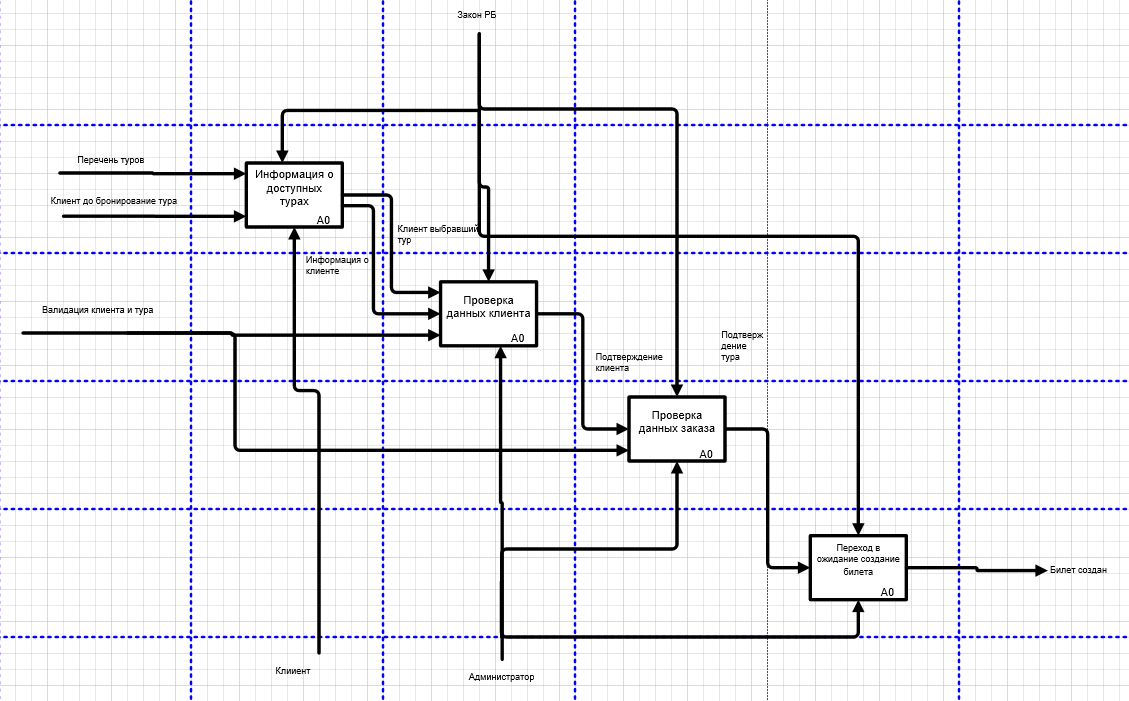


Рисунок А.4 - Декомпозиция процесса туристической фирмы

## ПРИЛОЖЕНИЕ Б

**(обязательное)**

**Информационная модель системы туристической фирмы**

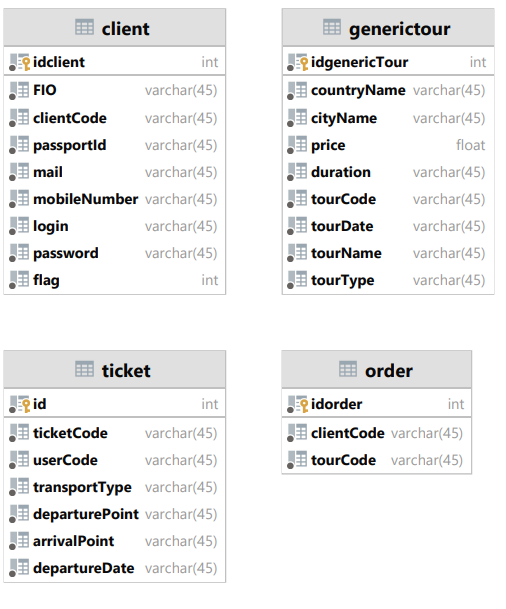


Рисунок Б.1 - Информационная модель системы управление туристической фирмы

## ПРИЛОЖЕНИЕ В

**(обязательное)**

**Диаграмма последовательности**

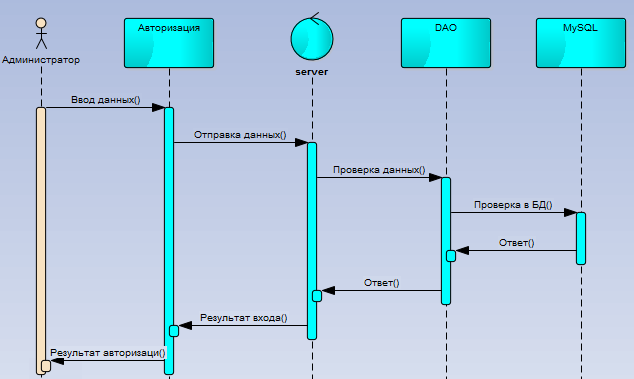


Рисунок Д.1 – Диаграмма последовательностей авторизации пользователя

## ПРИЛОЖЕНИЕ Г

**(обязательное)**

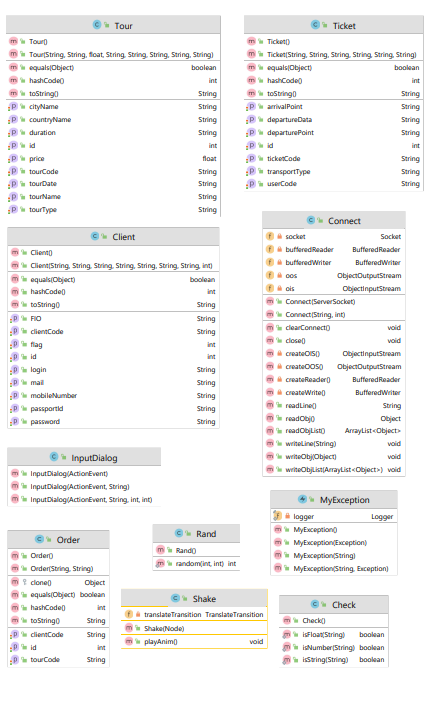
**Диаграмма состояний системы бронирование туристических услуг**

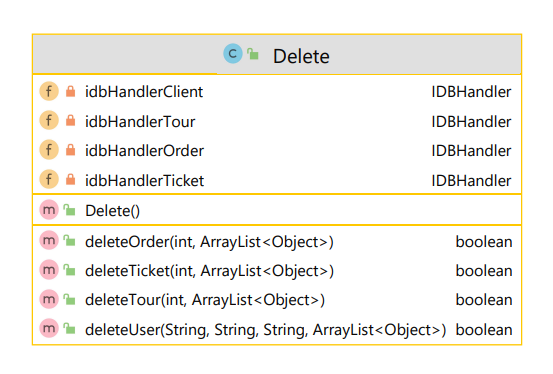
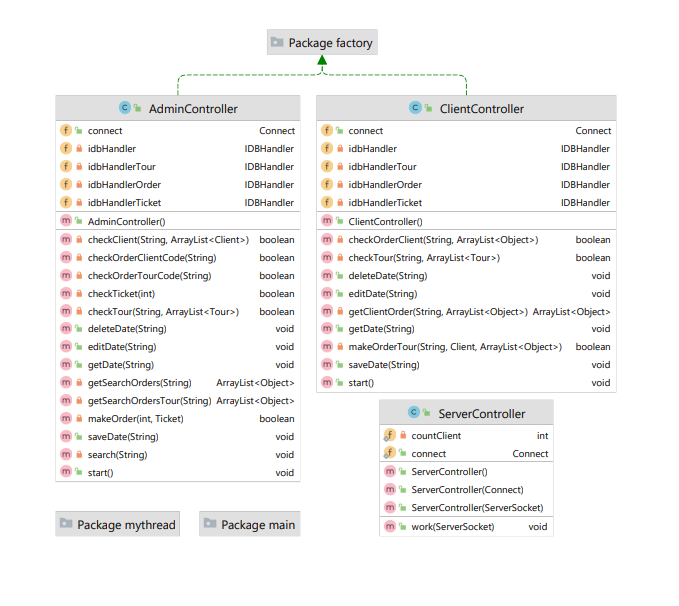
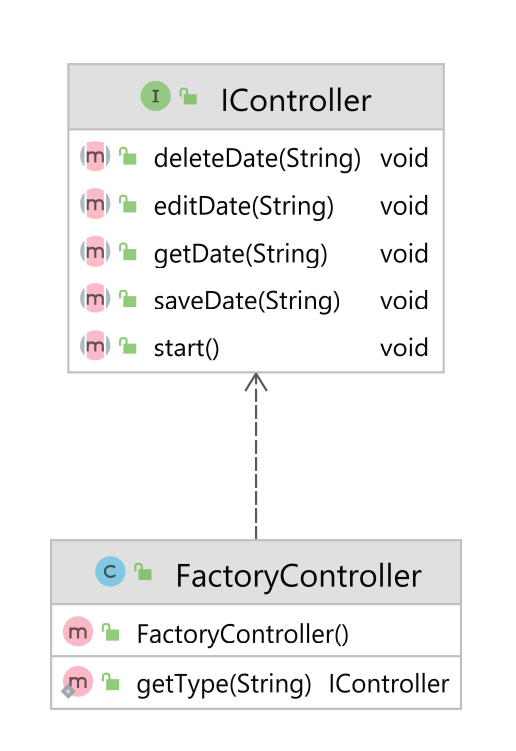
## 

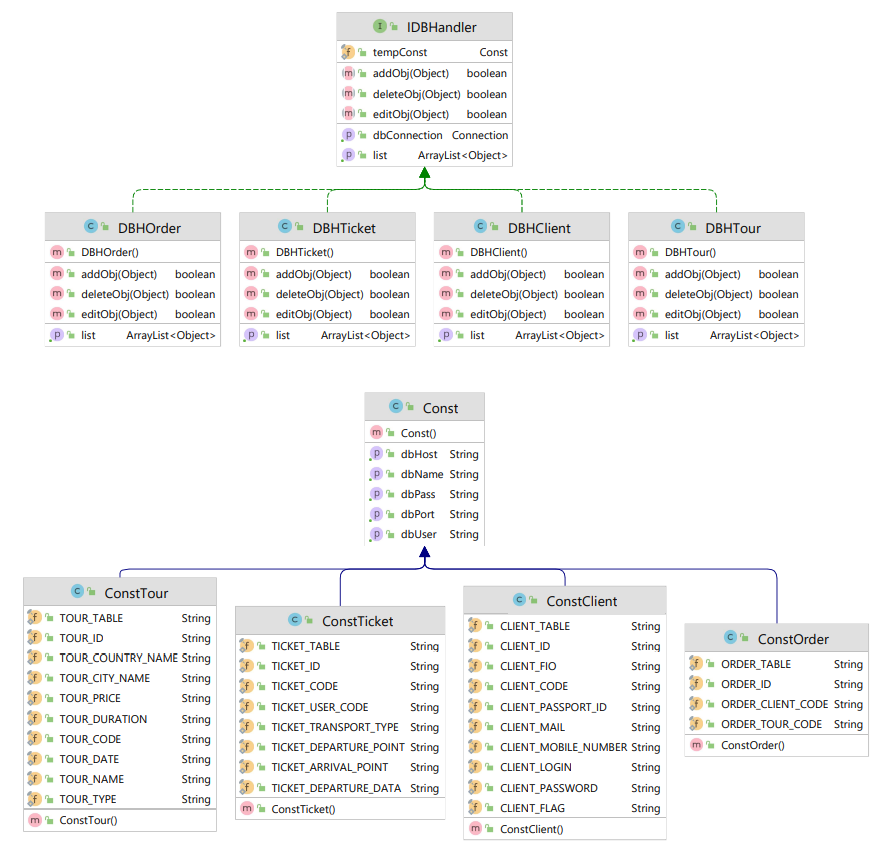
Рисунок Г.1 – Диаграмма состояний бронирование тура.

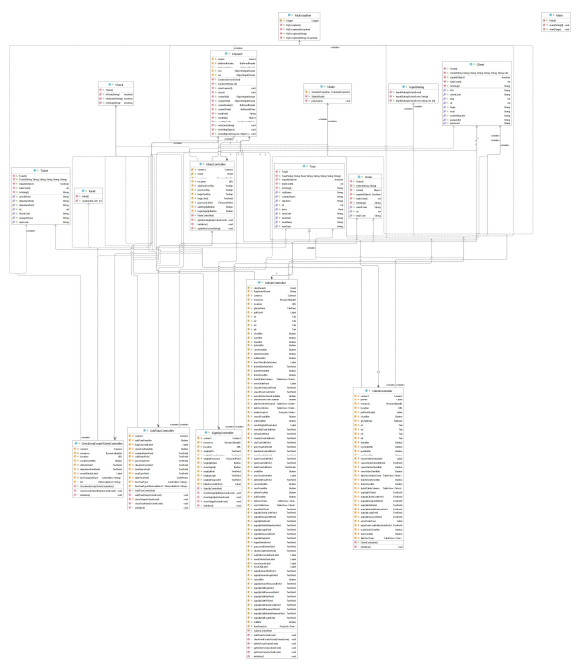
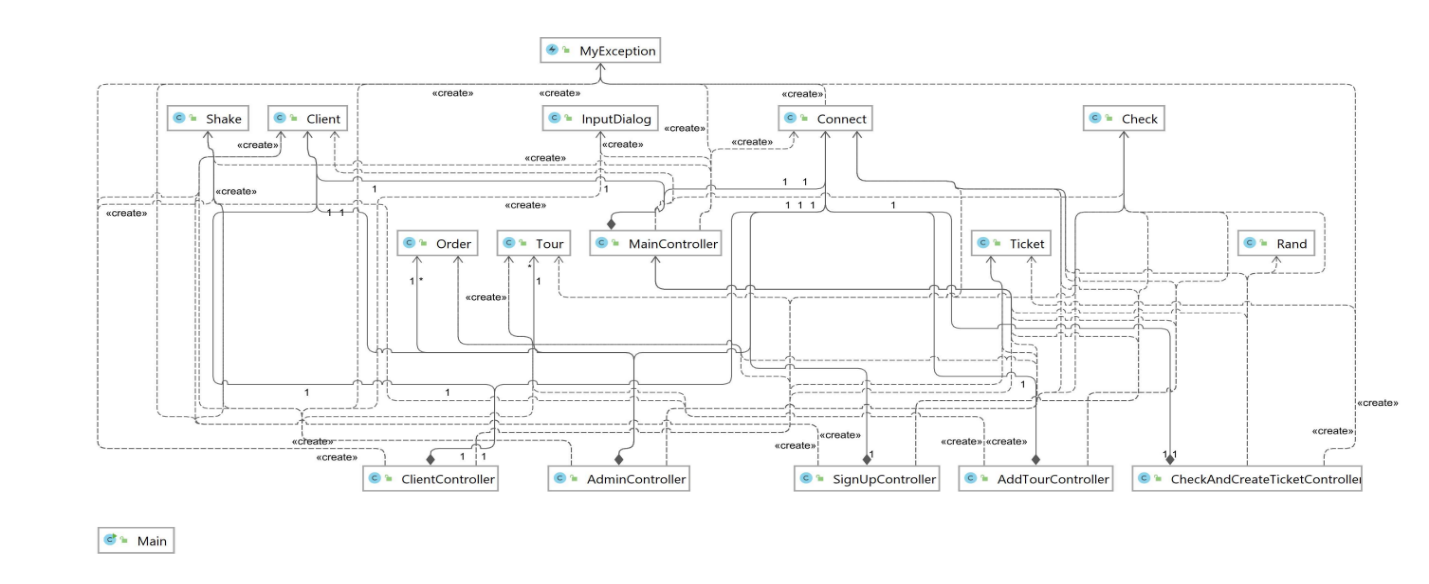
## ПРИЛОЖЕНИЕ Д

**(обязательное)**

**Диаграмма классов системы туристических услуг.**

Рисунок Д.1 – Диаграмма классов системы управления

 Рисунок Д.2 – Диаграмма классов пакета db

Рисунок Д.3 – Диаграмма классов на уровне пакета.

## ПРИЛОЖЕНИЕ Е

**(обязательное)**

Диаграмма вариантов использования системы управления туристических услуг, администратором, и клиентом.

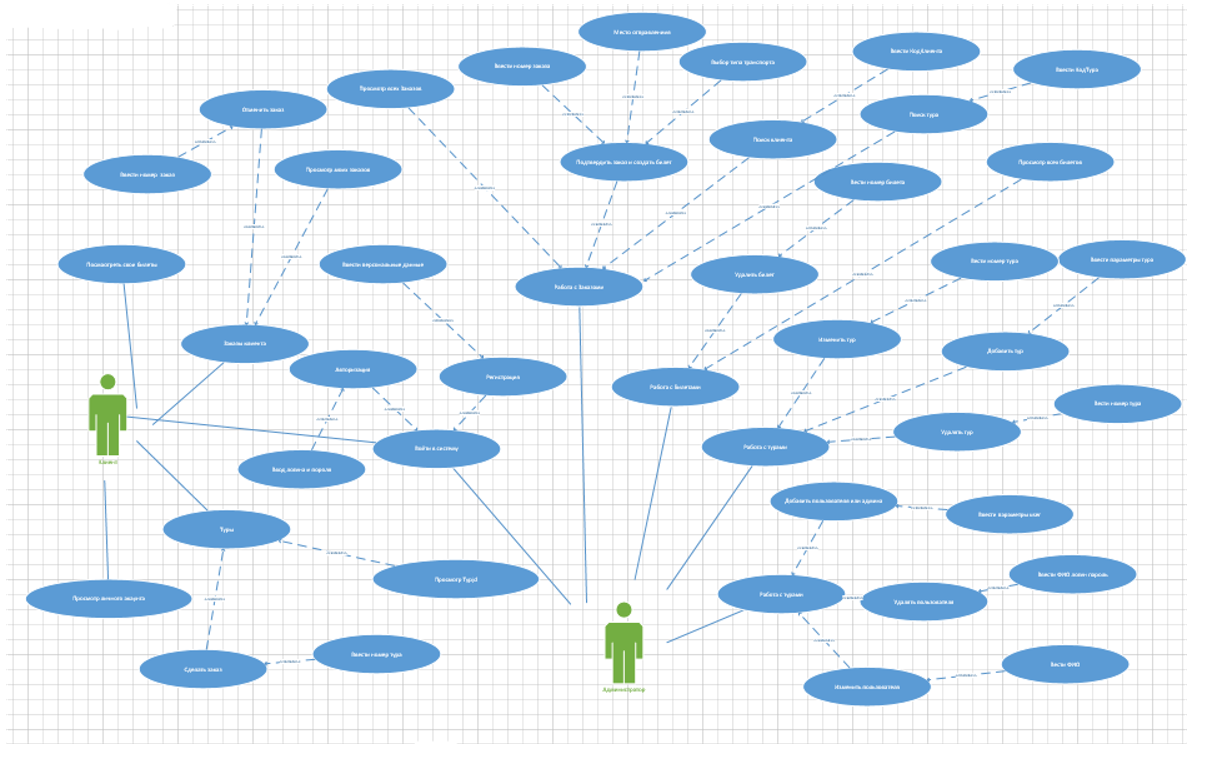


Рисунок Е.1 Диаграмма вариантов использования.

## ПРИЛОЖЕНИЕ Ж

**(обязательное)**

Скрипт базы данных туристической фирмы.

create table client

(

idclient int auto\_increment

primary key,

FIO varchar(45) not null,

clientCode varchar(45) not null,

passportId varchar(45) not null,

mail varchar(45) not null,

mobileNumber varchar(45) not null,

login varchar(45) not null,

password varchar(45) not null,

flag int not null

);

create table generictour

(

idgenericTour int auto\_increment

primary key,

countryName varchar(45) not null,

cityName varchar(45) not null,

price float not null,

duration varchar(45) not null,

tourCode varchar(45) not null,

tourDate varchar(45) not null,

tourName varchar(45) not null,

tourType varchar(45) not null

);

create table `order`

(

idorder int auto\_increment

primary key,

clientCode varchar(45) not null,

tourCode varchar(45) not null

);

create table ticket

(

id int auto\_increment

primary key,

ticketCode varchar(45) not null,

userCode varchar(45) not null,

transportType varchar(45) not null,

departurePoint varchar(45) not null,

arrivalPoint varchar(45) not null,

departureDate varchar(45) not null

);

## ПРИЛОЖЕНИЕ З

**(обязательное)**

package controller.factory;

import java.io.IOException;

public interface IController {

void saveDate(String msg) throws IOException, ClassNotFoundException;

void editDate(String msg) throws IOException, ClassNotFoundException;

void deleteDate(String msg) throws IOException, ClassNotFoundException;

void getDate(String msg) throws IOException, ClassNotFoundException;

void start();

}

package controller.factory;

import controller.AdminController;

import controller.ClientController;

/\*\*@author Ataeyv I.M.

\* Factory pattern\*/

public class FactoryController {

public static IController getType(String type) {

switch (type) {

case "admin":

return new AdminController();

case "client":

return new ClientController();

default:

throw new RuntimeException();

}

}

}

package controller.mythread;

import controller.ServerController;

import java.net.ServerSocket;

/\*\*@author Atayev I.M. \* \*/

public class MyThread extends Thread {

private Object locker;

private ServerSocket serverSocket;

public MyThread(ServerSocket serverSocket) {

this.serverSocket = serverSocket;

}

public MyThread(ServerSocket serverSocket,Object locker) {

this.serverSocket = serverSocket;

this.locker = locker;

}

@Override

public void run() {

synchronized (this.locker){

ServerController serverController = new ServerController();

serverController.work(serverSocket);

}

}

}

package model.bd.idbhandler;

import model.configs.constBD.Const;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.ArrayList;

public interface IDBHandler {

Const tempConst = new Const();

default Connection getDbConnection() throws SQLException, ClassNotFoundException {

String connectionString = "jdbc:mysql://" + tempConst.getDbHost() + ":" + tempConst.getDbPort() + "/" + tempConst.getDbName();

Class.forName("com.mysql.cj.jdbc.Driver");

Connection dbConnection = DriverManager.getConnection(connectionString, tempConst.getDbUser(), tempConst.getDbPass());

return dbConnection;

}

boolean addObj(Object obj);

ArrayList<Object> getList();

boolean deleteObj(Object obj);

boolean editObj(Object obj);

}

package model.configs.constBD;

public class Const {

protected final String dbHost = "localhost";

protected final String dbPort = "3306";

protected final String dbUser = "root";

protected final String dbPass = "admin123";

protected final String dbName = "tour";

public String getDbHost() {

return dbHost;

}

public String getDbPort() {

return dbPort;

}

public String getDbUser() {

return dbUser;

}

public String getDbPass() {

return dbPass;

}

public String getDbName() {

return dbName;

}

}

package com.example.model.tour;

import java.io.Serializable;

import java.util.Objects;

/\*\*@author Ataeyv I.M.

\* @implNote Serializable \* \*/

public class Tour implements Serializable {

private int id;

private String countryName;

private String cityName;

private float price;

private String duration;

private String tourCode;

private String tourDate;

private String tourName;

private String tourType;

public Tour() {

}

public Tour(String countryName, String cityName, float price, String duration,

String tourCode, String tourDate, String tourName, String tourType) {

this.countryName = countryName;

this.cityName = cityName;

this.price = price;

this.duration = duration;

this.tourCode = tourCode;

this.tourDate = tourDate;

this.tourName = tourName;

this.tourType = tourType;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getCountryName() {

return countryName;

}

public void setCountryName(String countryName) {

this.countryName = countryName;

}

public String getCityName() {

return cityName;

}

public void setCityName(String cityName) {

this.cityName = cityName;

}

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public String getDuration() {

return duration;

}

public void setDuration(String duration) {

this.duration = duration;

}

public String getTourCode() {

return tourCode;

}

public void setTourCode(String tourCode) {

this.tourCode = tourCode;

}

public String getTourDate() {

return tourDate;

}

public void setTourDate(String tourDate) {

this.tourDate = tourDate;

}

public String getTourName() {

return tourName;

}

public void setTourName(String tourName) {

this.tourName = tourName;

}

public String getTourType() {

return tourType;

}

public void setTourType(String tourType) {

this.tourType = tourType;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Tour tour = (Tour) o;

return Float.compare(tour.price, price) == 0

&& Objects.equals(countryName, tour.countryName)

&& Objects.equals(cityName, tour.cityName)

&& Objects.equals(duration, tour.duration)

&& Objects.equals(tourCode, tour.tourCode)

&& Objects.equals(tourDate, tour.tourDate)

&& Objects.equals(tourName, tour.tourName)

&& Objects.equals(tourType, tour.tourType);

}

@Override

public int hashCode() {

return Objects.hash(countryName, cityName, price, duration, tourCode, tourDate, tourName, tourType);

}

@Override

public String toString() {

return "Tour{" +

"id=" + id +

", countryName='" + countryName + '\'' +

", cityName='" + cityName + '\'' +

", price=" + price +

", duration='" + duration + '\'' +

", tourCode='" + tourCode + '\'' +

", tourDate='" + tourDate + '\'' +

", tourName='" + tourName + '\'' +

", tourType='" + tourType + '\'' +

'}';

}

}

package com.example.model.ticket;

import java.io.Serializable;

import java.util.Objects;

/\*\*@author Atayev I.M. \*/

public class Ticket implements Serializable {

private int id;

private String ticketCode;

private String userCode;

private String transportType;

private String departurePoint;

private String arrivalPoint;

private String departureData;

public Ticket() {

}

public Ticket(String ticketCode, String userCode, String transportType,

String departurePoint, String arrivalPoint, String departureData) {

this.ticketCode = ticketCode;

this.userCode = userCode;

this.transportType = transportType;

this.departurePoint = departurePoint;

this.arrivalPoint = arrivalPoint;

this.departureData = departureData;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getTicketCode() {

return ticketCode;

}

public void setTicketCode(String ticketCode) {

this.ticketCode = ticketCode;

}

public String getUserCode() {

return userCode;

}

public void setUserCode(String userCode) {

this.userCode = userCode;

}

public String getTransportType() {

return transportType;

}

public void setTransportType(String transportType) {

this.transportType = transportType;

}

public String getDeparturePoint() {

return departurePoint;

}

public void setDeparturePoint(String departurePoint) {

this.departurePoint = departurePoint;

}

public String getArrivalPoint() {

return arrivalPoint;

}

public void setArrivalPoint(String arrivalPoint) {

this.arrivalPoint = arrivalPoint;

}

public String getDepartureData() {

return departureData;

}

public void setDepartureData(String departureData) {

this.departureData = departureData;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Ticket ticket = (Ticket) o;

return Objects.equals(ticketCode, ticket.ticketCode)

&& Objects.equals(userCode, ticket.userCode)

&& Objects.equals(transportType, ticket.transportType)

&& Objects.equals(departurePoint, ticket.departurePoint)

&& Objects.equals(arrivalPoint, ticket.arrivalPoint)

&& Objects.equals(departureData, ticket.departureData);

}

@Override

public int hashCode() {

return Objects.hash(ticketCode, userCode, transportType, departurePoint, arrivalPoint, departureData);

}

@Override

public String toString() {

return "Ticket{" +

"id=" + id +

", ticketCode='" + ticketCode + '\'' +

", userCode='" + userCode + '\'' +

", transportType='" + transportType + '\'' +

", departurePoint='" + departurePoint + '\'' +

", arrivalPoint='" + arrivalPoint + '\'' +

", departureData='" + departureData + '\'' +

'}';

}

}

package com.example.model.order;

import java.io.Serializable;

import java.util.Objects;

/\*\*

\* @author Atayev I.M.

\* \*/

public class Order implements Serializable {

private int id;

private String clientCode;

private String tourCode;

public Order() {

}

public Order(String clientCode, String tourCode) {

this.clientCode = clientCode;

this.tourCode = tourCode;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getClientCode() {

return clientCode;

}

public void setClientCode(String clientCode) {

this.clientCode = clientCode;

}

public String getTourCode() {

return tourCode;

}

public void setTourCode(String tourCode) {

this.tourCode = tourCode;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Order order = (Order) o;

return Objects.equals(clientCode, order.clientCode) && Objects.equals(tourCode, order.tourCode);

}

@Override

public int hashCode() {

return Objects.hash(clientCode, tourCode);

}

@Override

public String toString() {

return "Order{" +

"id=" + id +

", clientCode='" + clientCode + '\'' +

", tourCode='" + tourCode + '\'' +

'}';

}

@Override

protected Object clone() throws CloneNotSupportedException {

return new Order(clientCode, tourCode);

}

}

package com.example.model.client;

import java.io.Serializable;

import java.util.Objects;

public class Client implements Serializable {

private int id;

private String FIO;

private String clientCode;

private String passportId;

private String mail;

private String mobileNumber;

private String login;

private String password;

private int flag;

public Client() {

}

public Client(String FIO, String clientCode, String passportId, String mail,

String mobileNumber, String login, String password, int flag) {

this.FIO = FIO;

this.clientCode = clientCode;

this.passportId = passportId;

this.mail = mail;

this.mobileNumber = mobileNumber;

this.login = login;

this.password = password;

this.flag = flag;

}

public String getFIO() {

return FIO;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public void setFIO(String FIO) {

this.FIO = FIO;

}

public String getClientCode() {

return clientCode;

}

public void setClientCode(String clientCode) {

this.clientCode = clientCode;

}

public String getPassportId() {

return passportId;

}

public void setPassportId(String passportId) {

this.passportId = passportId;

}

public String getMail() {

return mail;

}

public void setMail(String mail) {

this.mail = mail;

}

public String getMobileNumber() {

return mobileNumber;

}

public void setMobileNumber(String mobileNumber) {

this.mobileNumber = mobileNumber;

}

public String getLogin() {

return login;

}

public void setLogin(String login) {

this.login = login;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public int getFlag() {

return flag;

}

public void setFlag(int flag) {

this.flag = flag;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Client client = (Client) o;

return flag == client.flag

&& Objects.equals(FIO, client.FIO)

&& Objects.equals(clientCode, client.clientCode)

&& Objects.equals(passportId, client.passportId)

&& Objects.equals(mail, client.mail)

&& Objects.equals(mobileNumber, client.mobileNumber)

&& Objects.equals(login, client.login)

&& Objects.equals(password, client.password);

}

@Override

public int hashCode() {

return Objects.hash(FIO, clientCode, passportId, mail, mobileNumber, login, password, flag);

}

@Override

public String toString() {

return "Client{" +

"id=" + id +

", FIO='" + FIO + '\'' +

", clientCode='" + clientCode + '\'' +

", passportId='" + passportId + '\'' +

", mail='" + mail + '\'' +

", mobileNumber='" + mobileNumber + '\'' +

", login='" + login + '\'' +

", password='" + password + '\'' +

", flag=" + flag +

'}';

}

}

package com.example.model.connect;

import com.example.model.myexception.MyException;

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

/\*\*

\* @author Atayev.I.M

\* \*/

public class Connect implements Closeable {

private Socket socket;

private BufferedReader bufferedReader;

private BufferedWriter bufferedWriter;

private ObjectOutputStream oos;

private ObjectInputStream ois;

public Connect(String ip, int port) {

try {

this.socket = new Socket(ip, port);

this.bufferedReader = createReader();

this.bufferedWriter = createWrite();

this.oos = createOOS();

this.ois = createOIS();

} catch (IOException e) {

e.printStackTrace();

}

}

public Connect(ServerSocket serverSocket) {

try {

this.socket = serverSocket.accept();

this.bufferedReader = createReader();

this.bufferedWriter = createWrite();

this.oos = createOOS();

this.ois = createOIS();

} catch (IOException e) {

e.printStackTrace();

}

}

private BufferedReader createReader() throws IOException {

return new BufferedReader(new InputStreamReader(socket.getInputStream()));

}

private BufferedWriter createWrite() throws IOException {

return new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

}

private ObjectOutputStream createOOS() throws IOException {

return new ObjectOutputStream(socket.getOutputStream());

}

private ObjectInputStream createOIS() throws IOException {

return new ObjectInputStream(socket.getInputStream());

}

public void writeLine(String message) throws IOException {

bufferedWriter.write(message);

bufferedWriter.newLine();

bufferedWriter.flush();

}

public String readLine() throws IOException {

return bufferedReader.readLine();

}

public Object readObj() throws IOException, ClassNotFoundException {

return ois.readObject();

}

public void writeObj(Object obj) throws IOException {

oos.writeObject(obj);

}

public ArrayList<Object> readObjList() throws IOException, ClassNotFoundException {

return (ArrayList<Object>) ois.readObject();

}

public void writeObjList(ArrayList<Object> objList) throws IOException {

oos.writeObject(objList);

oos.flush();

}

public void clearConnect() throws IOException {

oos.flush();

bufferedWriter.newLine();

bufferedWriter.flush();

}

@Override

public void close() {

try {

bufferedReader.close();

bufferedWriter.close();

oos.close();

ois.close();

socket.close();

} catch (IOException e) {

new MyException("Class Connect ",e);

}

}

}

package com.example.model.rand;

/\*\*

\* @author Ataeyv I.M.

\* It is class generates random number.

\* \*/

public class Rand {

public static int random(int min, int max) {

return min + (int) (Math.random() \* ((max - min) + 1));

}

}

package com.example.model.myexception;

import org.apache.logging.log4j.Level;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

/\*\*

\* @author Ataev Ismayll

\* on class create logger for write error

\* логер должен создоватся в проекте 1 раз

\*/

public class MyException extends Exception {

private static final Logger logger = LogManager.getLogger(MyException.class);

public MyException() {

super();

}

public MyException(String msg) {

super();

logger.log(Level.ERROR, msg);

}

public MyException(Exception exception) {

super();

logger.log(Level.ERROR, exception.getMessage());

}

public MyException(String msg, Exception exception) {

super();

logger.log(Level.ERROR, msg + exception.getMessage());

}

}

package com.example.model.dialog;

import javafx.event.ActionEvent;

import javafx.fxml.FXMLLoader;

import javafx.scene.Node;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.IOException;

public class InputDialog {

public InputDialog(ActionEvent actionEvent) {

Stage stage = new Stage();

Parent root = null;

try {

root = FXMLLoader.load(getClass().getResource("sign-up-ui.fxml"));

} catch (IOException e) {

e.printStackTrace();

}

stage.setTitle("Новое окно");

stage.setScene(new Scene(root, 300, 300));

stage.initModality(Modality.WINDOW\_MODAL);

stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());

stage.show(); }

public InputDialog(ActionEvent actionEvent, String path) {

Stage stage = new Stage();

Parent root = null;

try {

root = FXMLLoader.load(getClass().getResource(path));

} catch (IOException e) {

e.printStackTrace();

}

stage.setTitle("Новое окно");

stage.setScene(new Scene(root, 300, 300));

stage.initModality(Modality.WINDOW\_MODAL);

stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());

stage.show();

}

public InputDialog(ActionEvent actionEvent, String path, int h, int w) {

Stage stage = new Stage();

Parent root = null;

try {

root = FXMLLoader.load(getClass().getResource(path));

} catch (IOException e) {

e.printStackTrace();

}

stage.setTitle("Новое окно");

stage.setScene(new Scene(root, h, w));

stage.initModality(Modality.WINDOW\_MODAL);

stage.initOwner(((Node) actionEvent.getSource()).getScene().getWindow());

stage.show();

}

}

package com.example.model.check;

/\*\*

\* @author Ataev Ismayll

\* class Check intended for check String and Integer.

\* All methods static.

\* I thing, what is use thread then will bad ideas or call methods for static. :-)

\* \*/

public class Check {

/\*\*

\* Check value na zero, null and number fort integer.

\*/

public static boolean isNumber(String number) {

return !number.equals("") && number != null && number.matches("\\d+?");

}

public static boolean isFloat(String fltStr) {

try {

Float.parseFloat(fltStr);

return true;

} catch (NumberFormatException ex) {

return false;

}

}

/\*\*Check value zero and null.

\* @param str \*/

public static boolean isString(String str) {

return !str.equals("") && str != null;

}

}

package com.example.model.animation;

import javafx.animation.TranslateTransition;

import javafx.scene.Node;

import javafx.util.Duration;

public class Shake {

private TranslateTransition translateTransition;

public Shake(Node node) {

translateTransition = new TranslateTransition(Duration.millis(80), node);

translateTransition.setFromX(0f);

translateTransition.setByX(10f);

translateTransition.setCycleCount(3);

translateTransition.setAutoReverse(true);

}

public void playAnim() {

translateTransition.playFromStart();

}

}

## ПРИЛОЖЕНИЕ И

**(обязательное)**

## ПРИЛОЖЕНИЕ К

**(обязательное)**

## ПРИЛОЖЕНИЕ Л

**(обязательное)**

## ПРИЛОЖЕНИЕ М

**(обязательное)**

## ПРИЛОЖЕНИЕ Н

**(обязательное)**