Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных технологий

Кафедра информационных систем и технологий

Дисциплина: Языки программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

АВТОМАТИЗИРОВАННОЕ РАБОЧЕЕ МЕСТО СЕКРЕТАРЯ

БГУИР КП 1-40 01 01 022 ПЗ

Студент: гр. 081073 Самойло А.А.

Руководитель: Листопадов С.А.

Минск 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В практической деятельности предприятий, учреждений и организаций нередки случаи, когда необходимо произвести закупки за наличный расчет на других предприятиях, в розничной торговле, у физических лиц или произвести расчеты с командированными работниками. В таких случаях обычно работнику выдаются наличные денежные средства для выполнения определенных действий по поручению организации. Так же выдаются деньги для предстоящих командировочных расходов, для оплаты хозяйственных расходов, для покупки продукции, для оплаты выполненных работ, оказанных услуг, а также на иные цели.

Целью данного курсового проекта является создание программы позволяющей автоматизировать рабочее место секретаря.

Проблема автоматизации производственных процессов и процессов управления как средства повышения производительности труда всегда являлась и остается актуальной. Необходимость автоматизации управления объясняется задачами облегчения труда управленческого персонала, и в частности, секретаря, сдерживанием, вызываемым развитием производства; усложнением производственных связей; увеличением объемов управленческой функции.

Важную роль играет задача соответствия технической базы управления аналогичной базе производства, в отношении которого производится автоматизация.

На современном этапе автоматизации управления производством наиболее перспективным является автоматизация планово-управленческих функций на базе персональных ЭВМ, установленных непосредственно на

рабочих местах специалистов. Эти системы получили широкое распространение в организационном управлении под названием автоматизированных рабочих мест (АРМ). Это позволит использовать систему людям, не имеющим специальных знаний в области программирования, и одновременно позволит дополнять систему по мере надобности.

Для достижения цели курсовой работы необходимо выполнить следующие задачи:

- изучить сферы деятельности организации и определить обязанности секретаря;
- проанализировать рабочее место секретаря, определить уровень автоматизации рабочего места, выявить недостатки;
- подготовить набор требований к программному обеспечению;
- проанализировать аналоги;
- спроектировать базу данных;
- определить необходимые средства разработки;
- разработать программное обеспечение;
- протестировать разработанное программное обеспечение;
- подготовить руководство пользователя.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предприятие нуждалось в программном продукте, который мог хранить информацию о командировках и расходах. Раньше эти данные хранились в excel таблице и постоянно изменялись. Секретарь часто редактировал данные, следил за тем, где сейчас находятся сотрудники и вручную создавал необходимые отчеты. Однако весь этот процесс занимает достаточно большой промежуток времени. Для автоматизации и упрощения работы секретаря в первую очередь необходимо определить основные функции секретаря:

- планирование командировок;
- редактирование данных о сотрудниках предприятия;
- редактирование данных о командировках;
- расчет стоимости командировок;
- обеспечение руководителя оперативной информацией о текущих командировках;
- создание и печать отчета о командировке.

Обладая необходимыми программными и техническими средствами, секретарь улучшает свою производительность труда, сокращает время обработки различных запросов. При достаточном уровне автоматизации

сокращается время, затраченное на обслуживание одного клиента, что очень важно при современном темпе жизни человека.

Основной задачей разрабатываемого программного средства является ускорение и упрощение процесса обработки данных. Секретарь должен без особого труда найти необходимую ему информацию. Для этого будет разработана система окон, между которыми можно будет переключаться для выполнения определенных действий.

Для хранения всей необходимой информации необходимо спроектировать БД, а также предусмотреть возможность расширения. В базе данных необходимо сохранять информацию о сотрудниках, билетах и командировках.

2 ФОРМУЛИРОВКА ЗАДАНИЯ

Необходимо разработать десктопное приложение с возможностью работы без подключения к интернету.

В программе должны быть реализованы следующие функции:

- добавление сотрудника, билета, суточных, командировки;
- редактирование информации о сотруднике, билете, суточных, командировки;
- удаление сотрудника, билета, суточных, командировки;
- поиск билетов по месту отправления/прибытия;
- поиск сотрудника по имени и фамилии;
- поиск суточных по стране;
- поиск командировки по названию;
- генерация отчета о командировке.

Помимо вышеперечисленных функций необходимо предусмотреть возможность расширения. В будущем секретарю может потребоваться

хранение дополнительной информации, а также выполнение других различных функций. Добавление нового функционала не должно сильно влиять на корректную работоспособность старого.

Таким образом разработано задание на проектирование приложения.

3 АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Аналогов в сети интернет немного и их основной недостаток в том, что они созданы для средних и крупных бизнесов, а также платные и как правило весьма дорогостоящие. Такое решение больше подходит для крупных компаний с большим количеством сотрудников. В связи с этим актуальность программного продукта для небольших компаний становится весьма высокой.

Рассмотрим несколько таких решений:

- Ozon Командировки;
- Aviasales.

Ozon в рамках своего сервиса для бронирования отелей и билетов Ozon.Travel запустил систему оформления командировок Ozon Командировки. Она позволит автоматизировать согласование рабочих

поездок с учётом специфики среднего и крупного бизнеса. Сотрудники смогут сами собрать «корзину» услуг: авиа- и железнодорожные билеты, отель, трансферы, визы и другие. При этом система позволяет определить «роли» для разных работников, установив для них лимиты на расходы.

Ключевые особенности Ozon Командировки:

- web-Приложение, все данные хранятся на серверах Ozon;
- у сотрудников есть роли и лимиты;
- доступно только для юридических лиц;
- у командировок есть статусы;
- документы формируются автоматически.

Главная особенность программы состоит в том, что командировки создаются сотрудниками и отправляются на согласование секретарю. Данное решение позволяет снизить загруженность секретаря, но это также добавляет немало неудобств сотрудникам.

К достоинствам можно отнести интерфейс. Главная страница Ozon Командировки изображена на рисунке 3.1. Данный интерфейс удобен и содержит всю необходимую информацию.

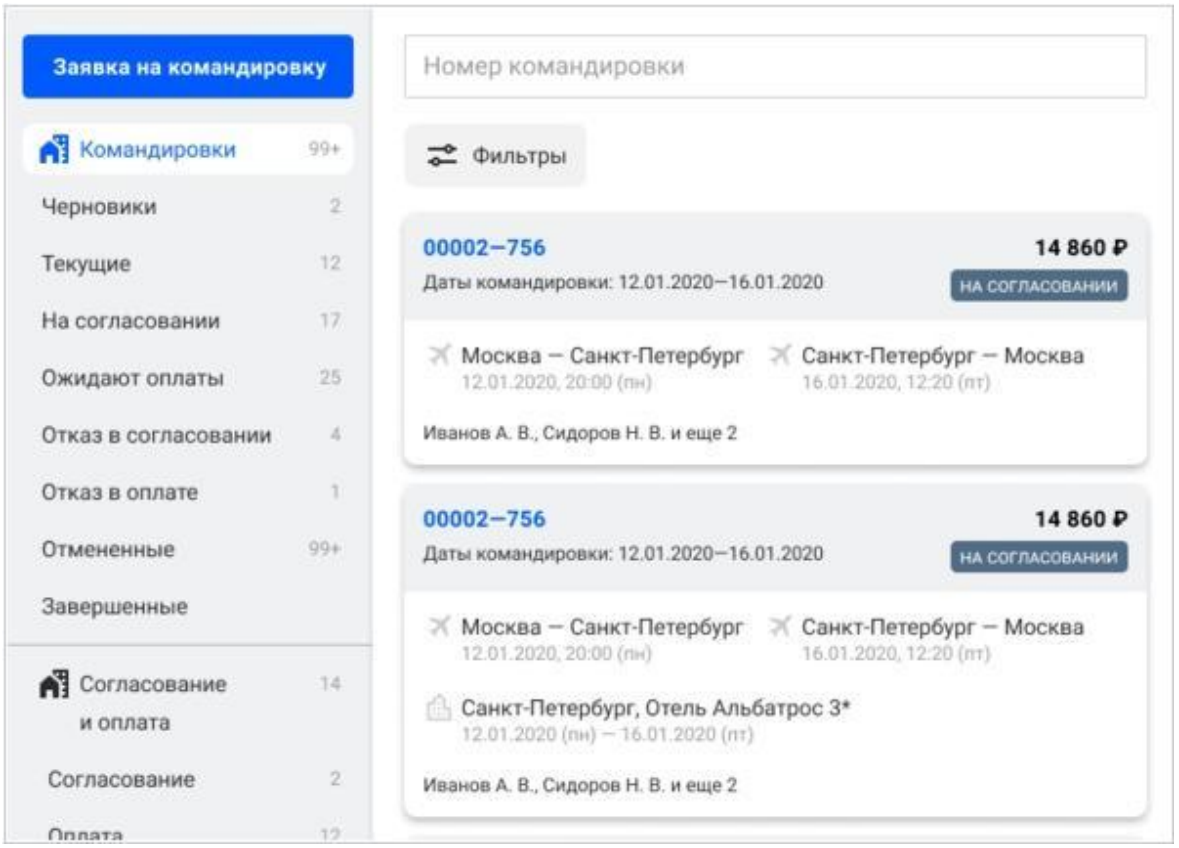

Рисунок 3.1 – Интерфейс Ozon Командировки

Недостаток Ozon Командировки состоит в том, что за использование приложения взимается плата, которая зависит как от количества сотрудников, так и от количества командировок. Так же к недостаткам можно отнести плохую совместимость с другими программными продуктами, например невозможно использовать единую базу данных сотрудников.

Сервис бронирования авиабилетов Aviasales решил выйти на корпоративный рынок и создал функционал для организации командировок. Он позволит вести учет командировок, создавать отчетные документы (документация будет приходить на почту бухгалтера и других ответственных сотрудников сразу после бронирования), осуществлять контроль и анализ расходов в поездках, настраивать политику бизнес-путешествий, производить оплату заказов с депозита и по постоплате.

Ключевые особенности Aviasales:

- web-Приложение, все данные хранятся на серверах Aviasales;
- быстрый поиск необходимых билетов;
- у командировок есть статусы;
- бесплатный сервис.

Главная особенность, а также и главный недостаток программы состоит в том, что командировки оформляются только с билетами купленными на Aviasales. Сервис предоставляет удобный поиск и покупку билетов, но отсутствует возможность указать дополнительную информацию о командировке. К примеру, невозможно указать дополнительные расходы на командировку.

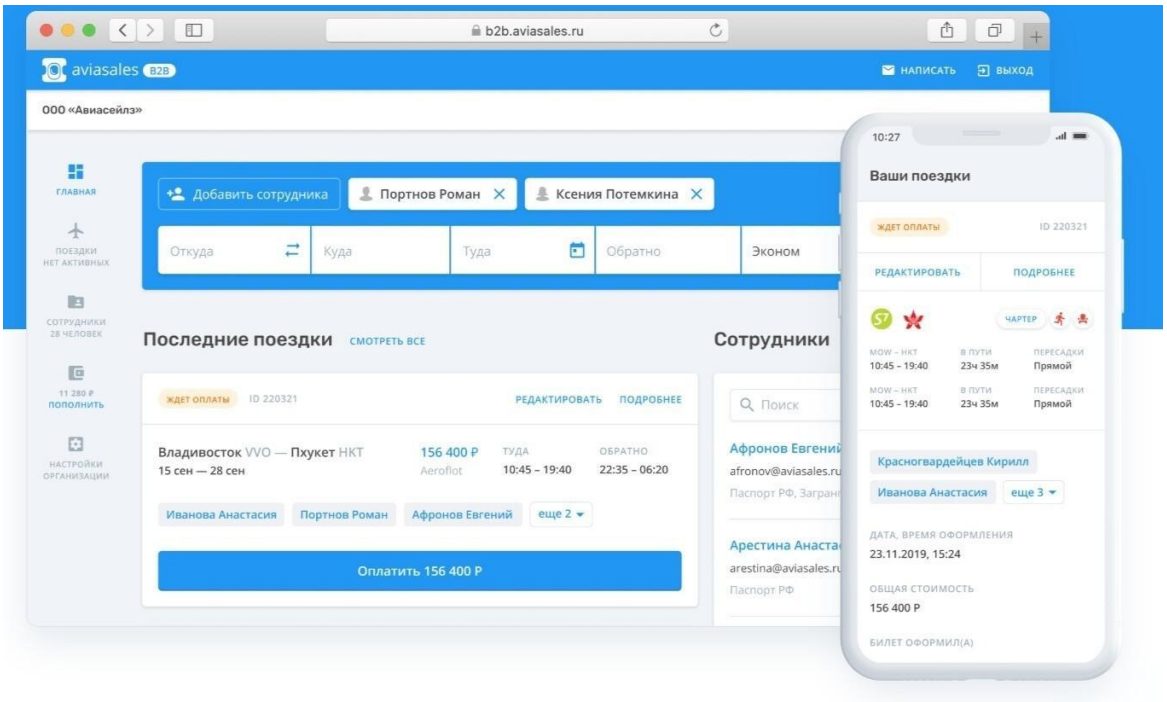

Рисунок 3.2 – Интерфейс Aviasales

На основании анализа существующих аналогов следует сделать вывод о том, что данные сервисы пользуются спросом как у малых, так и крупных бизнесов.

4 ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

В процессе логического моделирования необходимо определить последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата выполнения программного средства. Для этого можно использовать диаграмму вариантов использования.

Диаграмма вариантов использования предназначена для моделирования функциональных требований к системе (в виде сценариев взаимодействия пользователей с системой). Диаграммы вариантов использования показывают взаимодействия между вариантами использования и действующими лицами, отражая функциональные требования к системе с точки зрения пользователя.

Вариант использования представляет собой последовательность действий, выполняемых системой в ответ на событие, инициируемое некоторым пользователем (актером).

Рисунок 4.1 - Актёр

Вариант использования изображается на диаграмме эллипсом внутри, которого содержится его краткое название действия. Вариант использования описывает типичное взаимодействие между пользователем и системой.

Рисунок 4.2 — Вариант использования

Также есть еще один базовый элемент — операция отношения (Рисунок 4.3). Отношение — это семантическая связь между отдельными элементами модели. В нашем случае будем пользоваться только тремя из них:

- отношение ассоциации;
- отношение включения;
- отношение расширения.

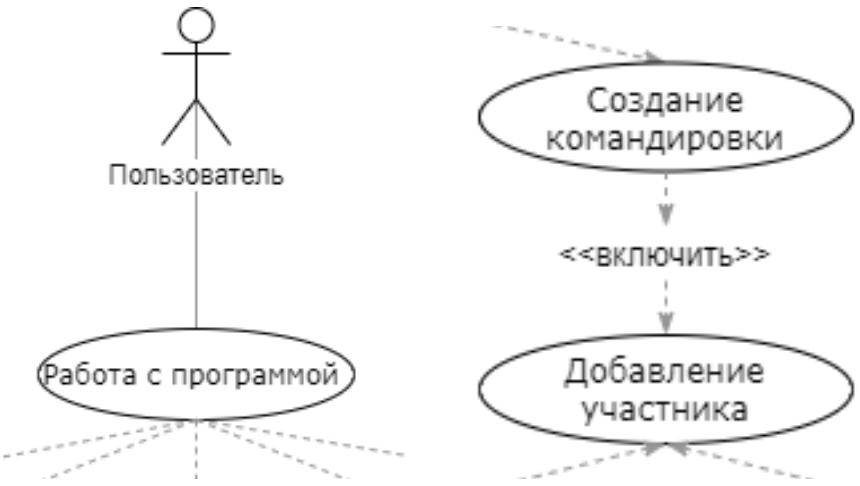

Рисунок 4.3 — Операции отношения (ассоциации, включения, расширения)

Диаграмма вариантов использования является самым общим представлением функциональных требований к системе. Полная диаграмма представлена на рисунке 4.4

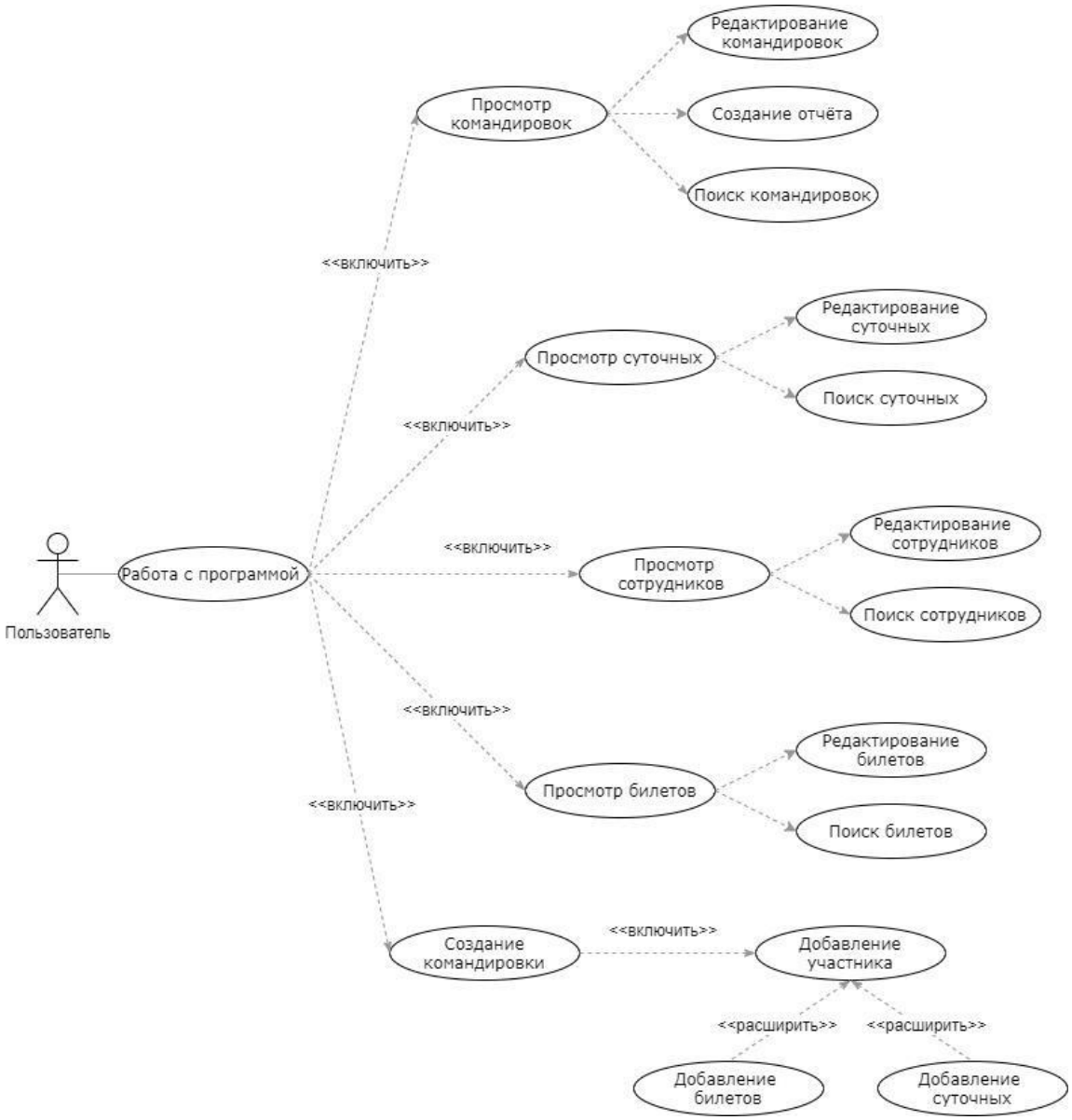

Рисунок 4.4 – Диаграмма вариантов использования

Помимо диаграммы вариантов использования необходимо разработать блок-схему (алгоритм) программы. Блок-схема представлена на рисунке 4.5.

Алгоритм — это точный набор инструкций, описывающих порядок действий некоторого исполнителя для достижения результата, решения некоторой задачи за конечное время.

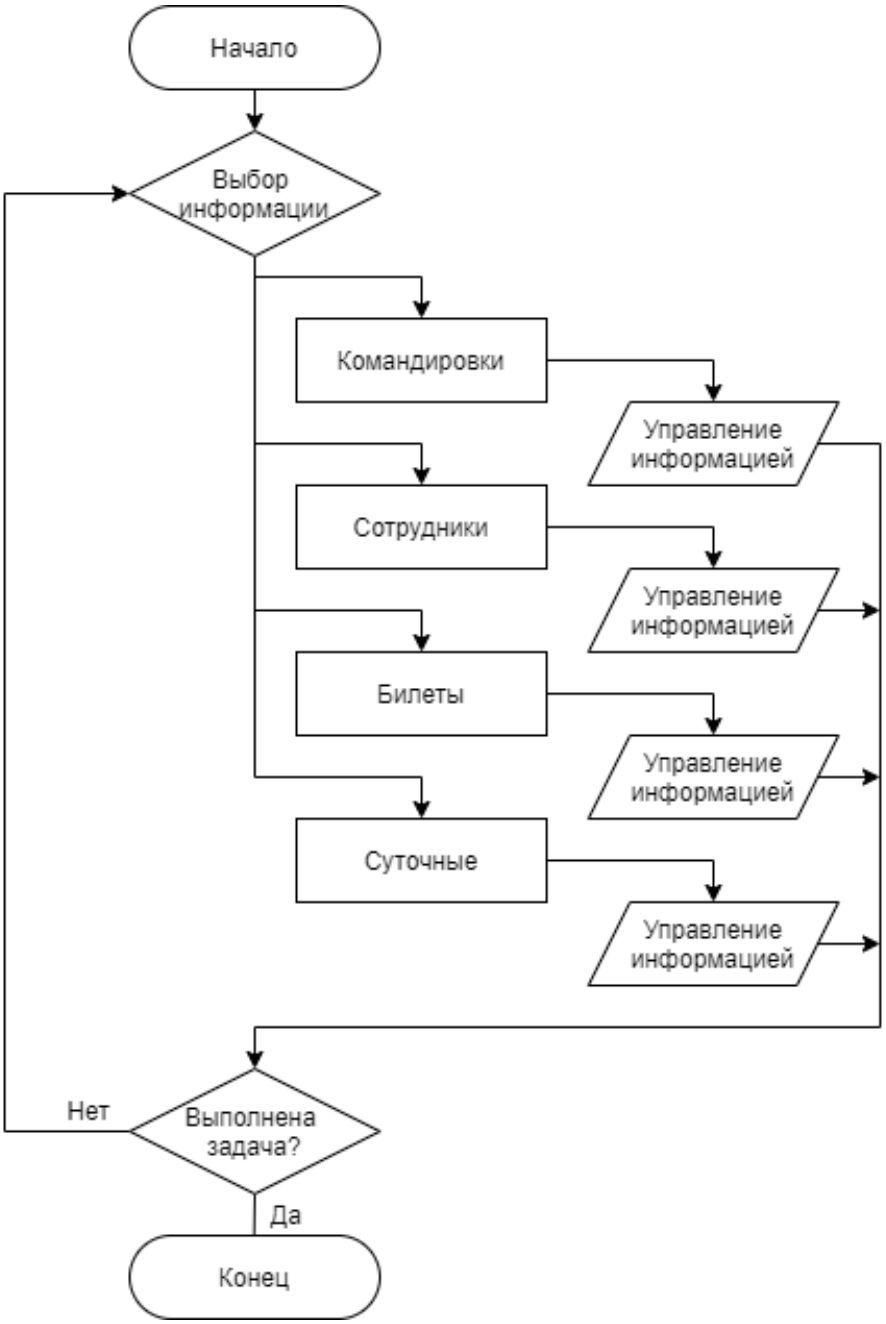

Рисунок 4.5 – Блок-схема программного средства

5 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

База данных (БД) — совокупность взаимосвязанных и организованных определенным образом данных.

Проектирование базы данных – процесс, который для заданного набора данных, относящихся к некоторой предметной области, позволяет выбрать и построить соответствующую оптимальную структуру.

База данных, используемая в разрабатываемом приложении, соответствует реляционной модели данных, где каждый выделенной в ходе проектировании сущности соответствует таблица. Структура базы данных разрабатываемого программного средства включает 6 таблиц. Структура данных и их краткое описание приводится в таблицах 5.1-5.6.

Схема базы данных разрабатываемого приложения представлена на рисунке 5.1.

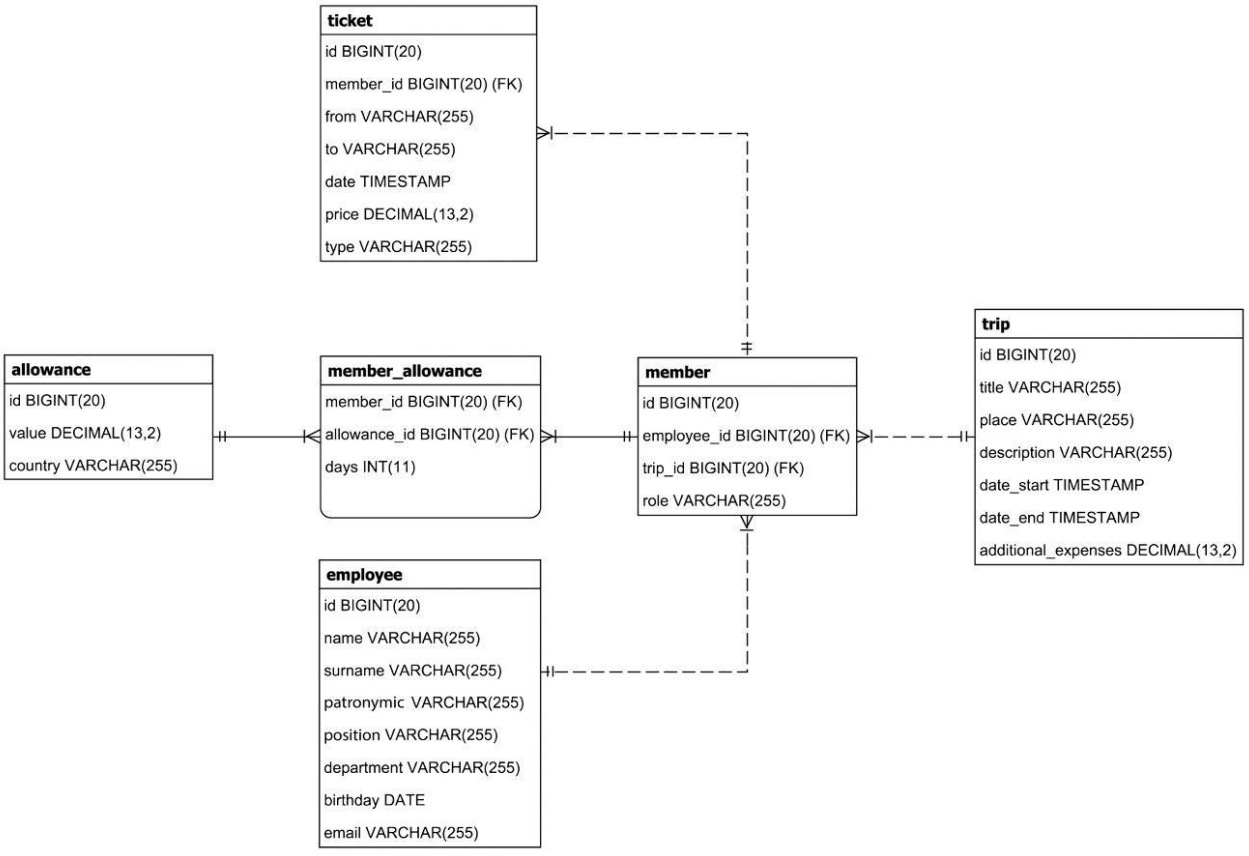

Рисунок 5.1 – Схема базы данных приложения

Таблица «employee» хранит информацию о сотрудниках компании. Структура данных приведена в таблице 5.1

Таблица 5.1 — Структура таблицы «employee»

Имя поля	Тип данных	Описание
id	bigserial	Идентификационный номер сотрудника
name	varchar	Имя сотрудника
surname	varchar	Фамилия сотрудника
patronymic	varchar	Отчество сотрудника
position	varchar	Занимаемая должность
department	varchar	Отдел, в котором работает сотрудник
birthday	date	Дата рождения сотрудника
email	varchar	Электронная почта сотрудника

Таблица «ticket» хранит информацию о билетах, купленных для участника командировки. Структура данных о билетах приведена в таблице 5.2

Таблица 5.2 — Структура таблицы «ticket»

Имя поля	Тип данных	Описание
id	bigserial	Идентификационный номер билета
member_id	int8	Код участника, на которого куплен билет
from	varchar	Страна (место) отправления
to	varchar	Страна (место) прибытия
date	timestamp	Дата отправления

price	float8	Цена билета
type	varchar	Тип билета (поезд, самолёт и т.д.)

Таблица «allowance» хранит информацию о суточных. Структура данных приведена в таблице 5.3

Таблица 5.3 — Структура таблицы «allowance»

Имя поля	Тип данных	Описание
id	bigserial	Идентификационный номер суточных
country	varchar	Место командировки
value	float8	Размер возмещения суточных

Таблица «member_allowance» хранит информацию о суточных для участника. Структура данных приведена в таблице 5.4

Таблица 5.4 — Структура таблицы «member_allowance»

Имя поля	Тип данных	Описание
allowance_id	int8	Код суточных
member_id	int8	Код участника
days	int4	Количество дней

Таблица «member» хранит информацию о участниках командировки. Структура данных приведена в таблице 5.5

Таблица 5.5 — Структура таблицы «member»

Имя поля	Тип данных	Описание
id	bigserial	Идентификационный номер участника
employee_id	int8	Код сотрудника

trip_id	int8	Код командировки, в которой участвует
role	varchar	Роль сотрудника в командировке

Таблица «trip» хранит информацию о командировках. Структура данных приведена в таблице 5.6

Таблица 5.6 — Структура таблицы «trip»

Имя поля	Тип данных	Описание
id	bigserial	Идентификационный номер командировки
title	varchar	Название (цель) командировки
description	varchar	Описание, цель, доп. информация о командировке
place	varchar	Событие, место
date_start	timestamp	Дата начала командировки
date_end	timestamp	Дата окончания командировки
additional_expensies	float8	Дополнительные расходы (на оборудование, покупки и т.д.)

6 РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ПРИЛОЖЕНИЯ

Пользовательский интерфейс будет оснащён всеми необходимыми функциями для работы секретаря. В качестве которых будет: поиск, добавление командировки, сотрудников, билетов и суточных, редактирование и удаление любой информации, создание отчета, а также другие функции, которые могут понадобиться.

В таблице 6.1 описаны все окна (формы) используемые в программе, а также, функции, которые они выполняют.

Таблица 6.1 – Функции форм

Форма	Функции
Главное окно	Просмотр сотрудников, которые были в командировках, а также сами командировки. Переход на другие формы.
Список всех сотрудников	На данной форме можно выбрать нужного сотрудника из списка или найти по названию, а также удалить и добавить нового.
Список всех командировок	На данной форме можно выбрать нужную командировку из списка или найти по фамилии и имени, а также удалить и создать отчёт.
Список всех билетов	На данной форме можно выбрать нужный билет из списка или найти по месту

	отправления или прибытия, а также удалить и добавить новый.
Список всех суточных	На данной форме можно выбрать суточные из списка или найти по стране, а также удалить и добавить новые.
Окно редактирования сотрудника	Просмотр и редактирование информации о сотруднике.
Окно редактирования командировки	Просмотр и редактирование информации о командировке.

Продолжение таблицы 6.1

Форма	Функции
Окно редактирования билета	Просмотр и редактирование информации о билете.
Окно редактирования суточных	Просмотр и редактирование информации о суточных.
Окно добавления командировки	На данной форме заполняются все необходимые данные для создания новой командировки.
Окно добавление участника	На данной форме заполняются все необходимые данные для участника командировки.
Окно добавления суточных	Данная форма необходима для добавления суточных для участника.
Окно с сообщением об ошибке	Данная форма сообщает пользователю, что операция не была выполнена.

На рисунке 6.1 представлен шаблон главного окна программы, на нём присутствуют элементы управления с помощью которых можно открывать новые окна, просматривать сотрудников и командировки.

Рисунок 6.1 – Шаблон главного окна

На рисунке 6.2 представлен шаблон окна, на котором будет представлен список сотрудников, командировок, билетов или суточных в зависимости от выбранной категории.

Рисунок 6.2 – Шаблон окна со списком объектов

На рисунках 6.3 – 6.6 представлены шаблоны окон для редактирования информации.

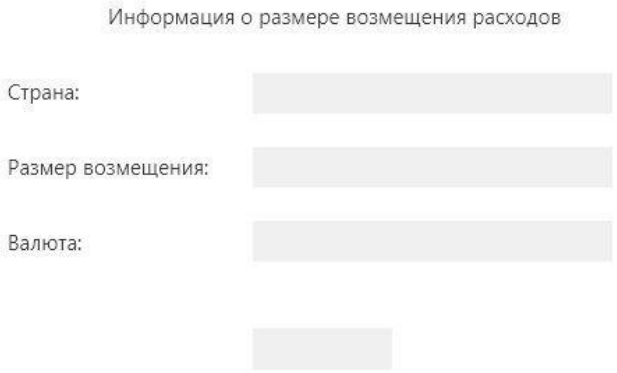

Рисунок 6.3 – Шаблон окна редактирования информации о суточных

Рисунок 6.4 – Шаблон окна редактирования информации о сотруднике

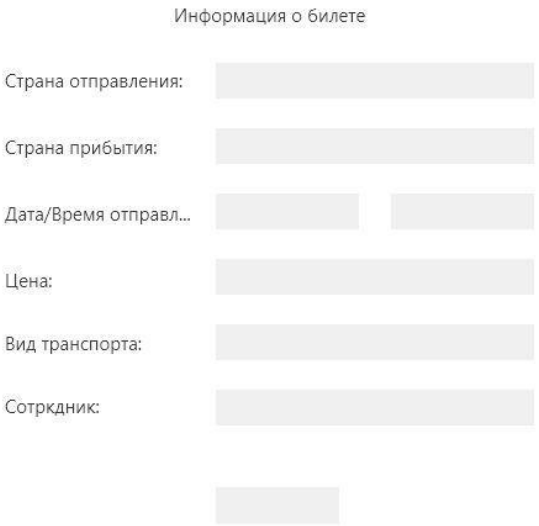

Рисунок 6.5 – Шаблон окна редактирования информации о билете

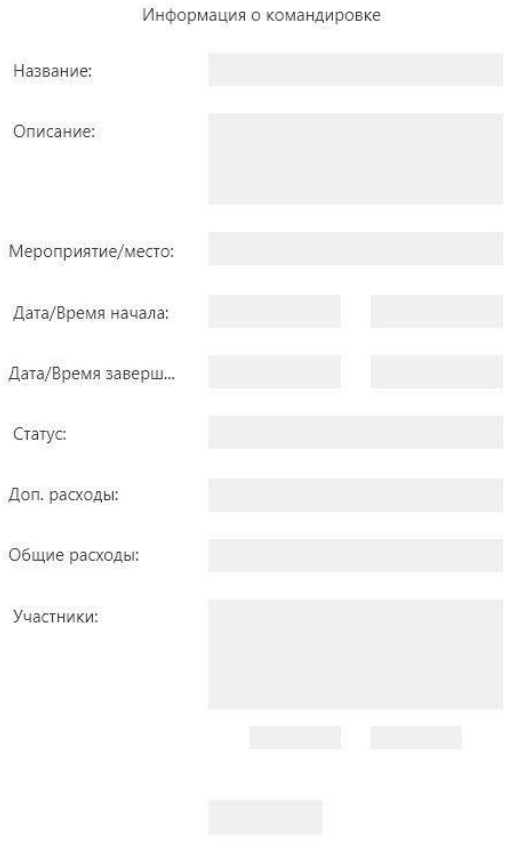

Рисунок 6.6 – Шаблон окна редактирования информации о командировке

7 ФИЗИЧЕСКОЕ МОДЕЛИРОВАНИЕ

7.1 Обоснование выбора среды и языка программирования

Разрабатываемый продукт будет разрабатывается с помощью операционной системы Microsoft Windows 10.

Microsoft Windows 10 обладает множеством преимуществ над своими младшими версиями такими, как визуальное оформление, быстродействие, поддержка новых технологий и тому подобное, что позволяет не только разрабатывать программный продукт в полном соответствии со всеми заданными условиями, но и создать удобный и интуитивно понятный интерфейс программы, что играет немаловажную роль в проектировании информационно-справочных систем.

Программное обеспечение будет написано на java.

В java 8 включены все необходимые компоненты и библиотеки для создания полноценного приложения. Основой для приложения послужит фреймворк Spring boot, для интерфейса будет использоваться фреймворк javafx.

Запросы для манипулирования данными будут написаны на JPQL. JPQL — это объектно-ориентированный язык запросов, его особенностью является абстрагирование от структуры БД, запросы создаются на основе java объектов, и с помощью фреймворка Hibernate автоматически преобразуются в SQL.

Для сборки проекта будет использоваться maven. Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в POM файлах.

В качестве среды разработки приложения будет использоваться IntelliJ IDEA. IntelliJ IDEA – это умная и удобная среда разработки для Java, включающая поддержку всех последних технологий и фреймворков. Она предоставляет инструменты для продуктивной работы и идеально подходит для создания коммерческих, мобильных и веб-приложений.

7.2 Критерии выбора СУБД

В качестве базы данных будет использоваться PostgreSQL. Основанием для выбора данной базы данных послужило то, что PostgreSQL не просто реляционная, а объектно-реляционная СУБД что даёт возможность создания пользовательских объектов и их поведения, включая типы данных, функции, сиквенсы и индексы. В данном программном обеспечении будут использоваться сиквенсы для идентификатора записи. Сиквенс (sequence) – это объект базы данных, который генерирует целые числа в соответствии с правилами, установленными во время его создания.

Также будет возможность запуска приложения с встроенной H2 базой данных. H2 — это легковесная база данных для Java с открытым исходным кодом. Данная БД в основном используется для разработки и тестирования. H2 позволит запустить приложение без установки дополнительных инструментов.

8 Реализация программного средства

8.1 Тестирование и отладка

Тестирование ПО — это процесс его исследования с целью получения информации о качестве. Целью тестирования является выявление дефектов в ПО. С помощью тестирования нельзя доказать отсутствие дефектов и корректность функционирования анализируемой программы. Тестирование

сложных программных продуктов является творческим процессом, не сводящимся к следованию строгим и четким процедурам.

Существует большое количество различных видов тестирования. Рассмотрим два основных:

Модульное тестирование. Каждая сложная программная система состоит из отдельных частей - модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе всей системы, необходимо сначала протестировать каждый модуль системы по отдельности. В случае возникновения проблем при тестировании системы в целом это позволяет проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них.

В Java за модульное тестирование отвечают специальные Test классы, которые работают с помощью Junit 5. JUnit — это библиотека для модульного тестирования программного обеспечения на языке Java. Принцип работы данной библиотеки заключается в сравнении ожидаемого результата с полученным. Тестирование запускается при компиляции кода чтобы обеспечить работоспособность готового приложения.

С помощью данной библиотеки будет осуществлено тестирование некоторых функций приложения т.к. конвертация строки с временем в java объект Date и конвертация сущности из БД в DTO. DTO (data transfer object) в переводе объект передачи данных - это один из шаблонов проектирования, используется для передачи данных между подсистемами приложения.

На рисунке 8.1 приведен пример тестирования конвертации объекта LocalDate (содержит только дату) и строки со временем в объект Date который содержит как дату, так и время и удобно подходит для записи в БД. На рисунке 8.2 изображен результат тестирования метода.

```
private final static int YEAR = 2000;
private final static int MONTH = 2;
private final static int DATE = 22;
private final static int HOUR = 22;
private final static int MINUTE = 22;
private final static int SECOND = 0;
private final static int MILLISECOND = 0;

@Test
void toDate() {
    Date expected = getTestDate();
    Date actual = ControllerUtil.toDate(LocalDate.of(YEAR, MONTH, DATE),  time: "22:22");
    assertEquals(expected, actual);
}

private static Date getTestDate() {
    Calendar calendar = Calendar.getInstance();
    calendar.set(YEAR,  month: MONTH - 1, DATE, HOUR, MINUTE, SECOND);
    calendar.set(Calendar.MILLISECOND, MILLISECOND);
    return calendar.getTime();
}
```

Рисунок 8.1– Тестирование конвертации даты

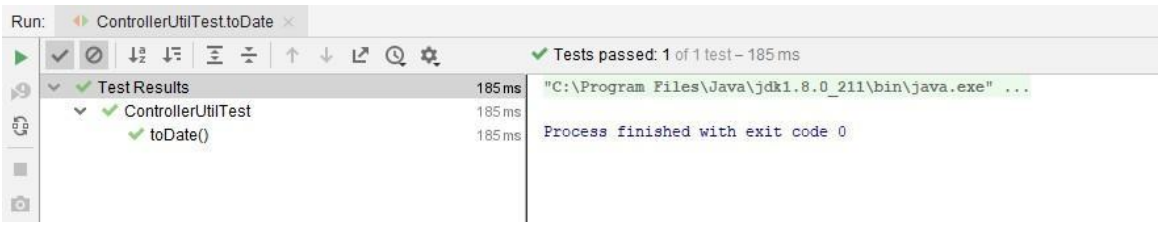

Рисунок 8.2 – Результат тестирования конвертации даты

На рисунке 8.3 приведен пример тестирования конвертации объекта Employee в EmployeDto. Данные объекты отличаются тем, что в DTO отсутствует огромное количество бесполезной информации о командировках, а содержится только её название и идентификатор. На рисунке 8.4 изображен результат тестирования метода.

```
private final static String NAME = "Test";
private final static String SURNAME = "Tester";
private final static String EMAIL = "test@test.ts";
private final static String DEPARTMENT = "Test";

@Test
void toDto() {
    Employee test = getTestEntity( id: 1L);
    EmployeeDto expected = new EmployeeDto( name: NAME + 1,  surname: SURNAME + 1,  email: EMAIL + 1,  department: DEPARTMENT + 1,  birthday: null);
    expected.setId(1L);
    EmployeeDto actual = new EmployeeServiceImpl().toDto(test);
    assertEquals(expected, actual);
}
```

Рисунок 8.3 – Тестирования конвертации объекта

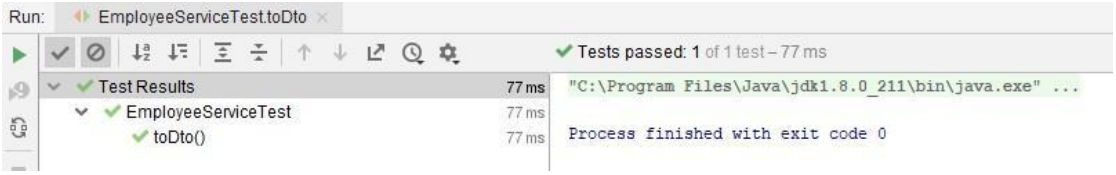

Рисунок 8.4 – Результат тестирования конвертации объекта

Функциональное тестирование. тестирование функций приложения на соответствие требованиям. Оценка производится в соответствии с ожидаемыми и полученными результатами (на основании функциональной спецификации), при условии, что функции отрабатывали на различных значениях. Функциональное тестирование будет представлено в разделе 8.2 «Разработка спецификации требований».

8.2 Разработка спецификации требований

В данном разделе представлены тест-кейсы основных действий пользователя при работе с программой.

Тест-кейс – это описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определённой ситуации, определённых выходных данных.

В Таблице 8.1 представлено тестирование основных действий пользователя на главном окне, ожидаемый результат и полученный.

Таблица 8.1 – Тестирование функций главного окна

Тест-кейс	Предусловие	Ход выполнения	Ожидаемый результат
Нажатие на сотрудника в таблице	Необходимы сотрудники и командировки в бд.	Нажать на сотрудника в таблице.	Отобразятся командировки, в которых участвовал выбранный сотрудник.
Нажатие на командировку в таблице	Необходимы сотрудники и командировки в бд.	Нажать на командировку в таблице.	Отобразятся участники выбранной командировки.
Показать все	-	Нажать на кнопку (Показать все) для командировок.	Переход на окно со списком командировок.
		Нажать на кнопку (Показать все) для сотрудников.	Переход на окно со списком сотрудников.
		Нажать на кнопку (Показать все билеты).	Переход на окно со списком билетов.
		Нажать на кнопку (Показать суточные).	Переход на окно со списком суточных.
Обновить	-	Нажать на кнопку (Обновить) командировками и сотрудниками.	Обновление данных в таблице с командировками и сотрудниками.
Добавить	-	Нажатие на кнопку (Добавить).	Переход на окно для добавления командировки.

В Таблице 8.2 приведены результаты тестирования всех возможных действий пользователя на окне со списком сотрудников и командировок. Также данное окно отвечает за вывод билетов и суточных. В зависимости от типа окна будут доступны различные действия, но они работают по одному и тому же принципу.

Таблица 8.2 – Тестирование функций окна со списком

Тест-кейс	Предусловие	Ход выполнения	Ожидаемый результат
Поиск	Открыть окно со списком сотрудников. Необходимы сотрудники с именем «Алексей» в бд.	Вводим “Алексей” в поле поиска.	Отобразился список сотрудников с именем «Алексей».
	Открыть окно со списком командировок. Необходимы командировки с названием «Тест» в бд.	Вводим «Тест» в поле поиска.	Отобразился список командировок с названием «Тест».
Добавить	Открыть окно не со списком командировок.	Нажатие на кнопку (Добавить).	Открылось окно для добавления.
	Открыть окно со списком командировок.	Нажатие на кнопку (Добавить).	Кнопка недоступна. Добавление командировок доступно только через главное окно.
Удаление	Не пустой список.	Выделить объект и нажать на (Удалить)	Выбранный объект удалён.
Отчёт	Открыть окно не со списком командировок.	Нажать на кнопку (Отчет)	Кнопка недоступна. Так как она доступна только при выводе командировок.

В Таблице 8.3 представлено тестирование всех действий пользователя на окнах редактирования информации о сотруднике. Все окна данного типа работают по одному и тому же принципу: заполнение информации, проверка информации, вывод окна об ошибке или сохранение изменений.

Таблица 8.3 – Тестирование функций окна редактирования сотрудника

Тест-кейс	Предусловие	Ход выполнения	Ожидаемый результат
Добавление сотрудника	Открыть окно добавления сотрудника. Корректно заполнить все поля.	Нажать на кнопку подтверждения.	Измененные данные сохранены, окно закрылось.
	Открыть окно добавления сотрудника. Не заполнять поля.	Нажать на кнопку подтверждения.	Выведено окно с сообщением об ошибке.

8.3 Руководство пользователя

8.3.1 Требования

Так как данная программа была написана на языке программирования java, то для ее функционирования требуется JVM (java virtual machine) в переводе виртуальная машина java. У JVM есть свои требования к системе, т.к. они не столь высоки, это позволит запускать программу на любых компьютерах независимо от операционной системы.

Для нормального функционирования программы JVM требует 126 МБ внешней памяти, 128 МБ оперативной памяти и процессор с частотой не

ниже 2 266 МГц. Так же необходимо минимум 40 Мб для программы. И т.к. БД может быть удалённой, то место под неё выделять не нужно.

Для работы с интерфейсом программы требуется монитор, компьютерная мышь и клавиатура. Так как программа не использует интернет то для её работы не требуется подключение к интернету.

8.3.2 Установка программы

Перед установкой убедитесь, что ваш компьютер удовлетворяет всем условиям, описанным в пункте 8.3.1.

Процесс установки достаточно прост, у конечного пользователя не должно возникнуть проблем с установкой программы. Пример установки изображен на рисунках 8.5 - 8.9.

Рекомендуется не менять стандартный путь установки, изображенный на рисунке 8.9. Можно создать ярлык на рабочем столе.

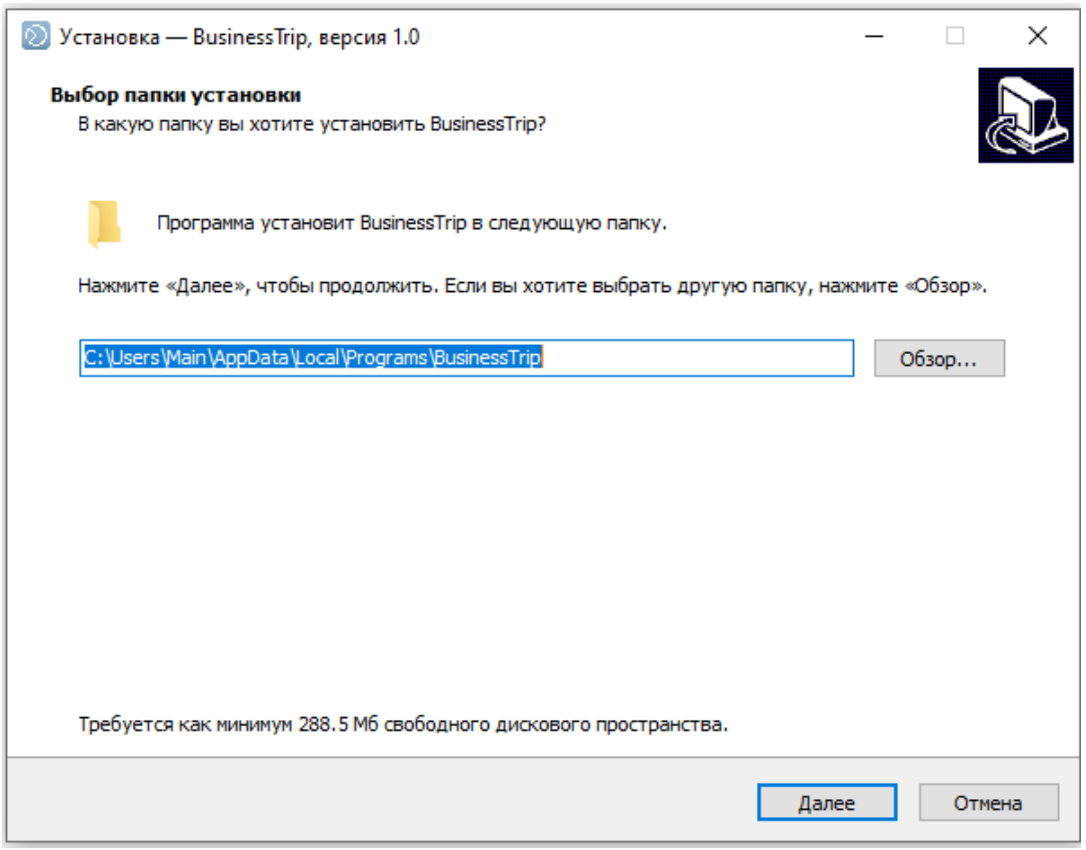

Рисунок 8.5 – Установка программы, выбор папки.

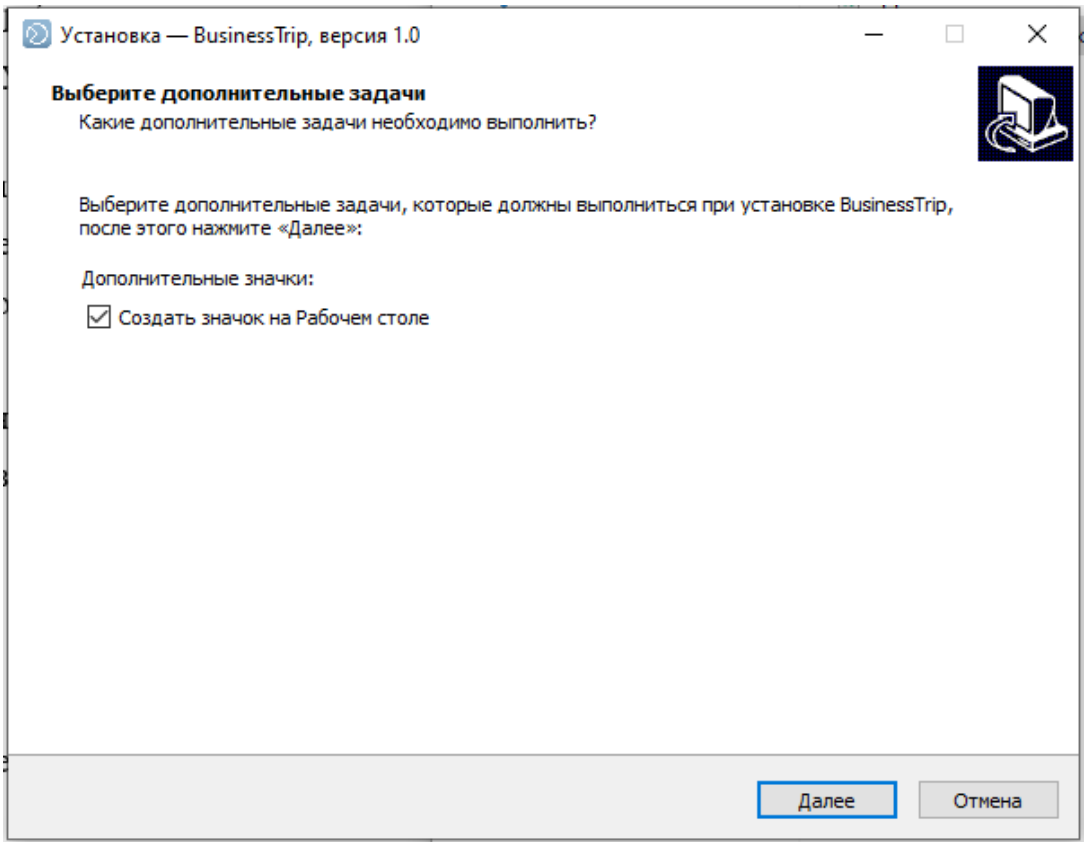

Рисунок 8.6 – Установка программы, дополнительные задачи.

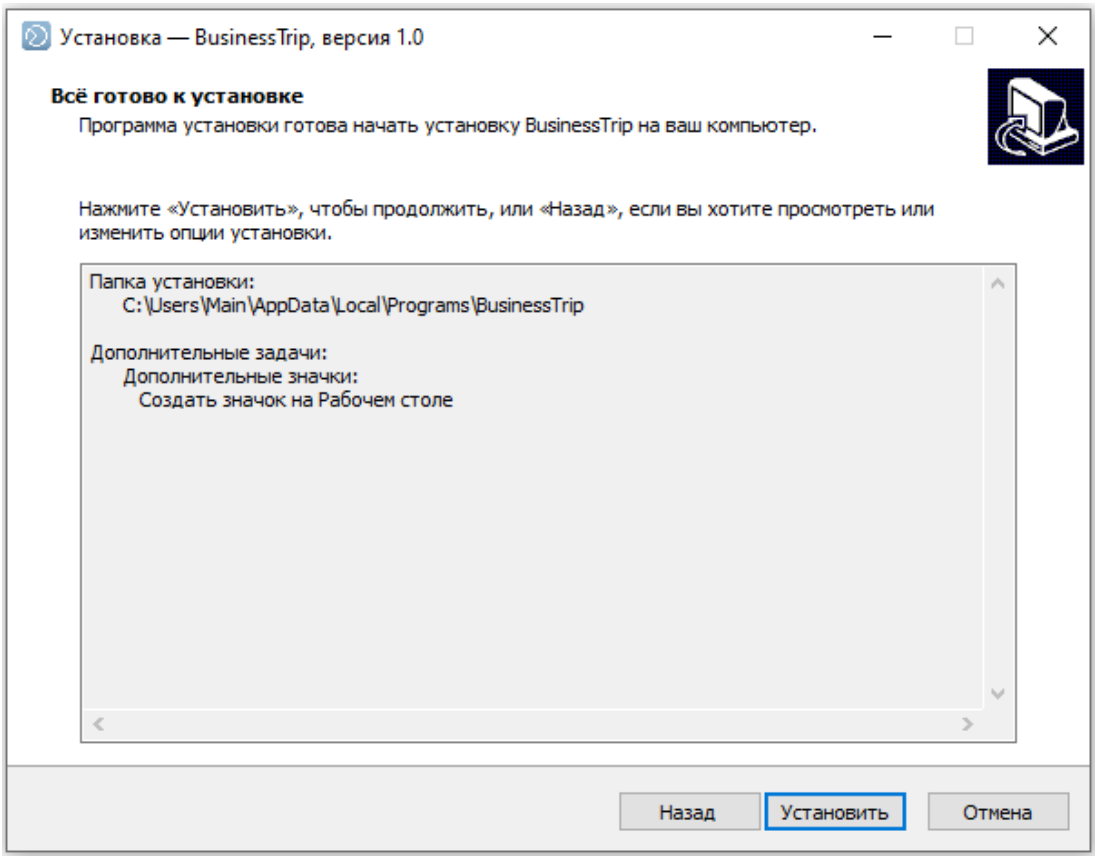

Рисунок 8.7 – Установка программы, проверка.

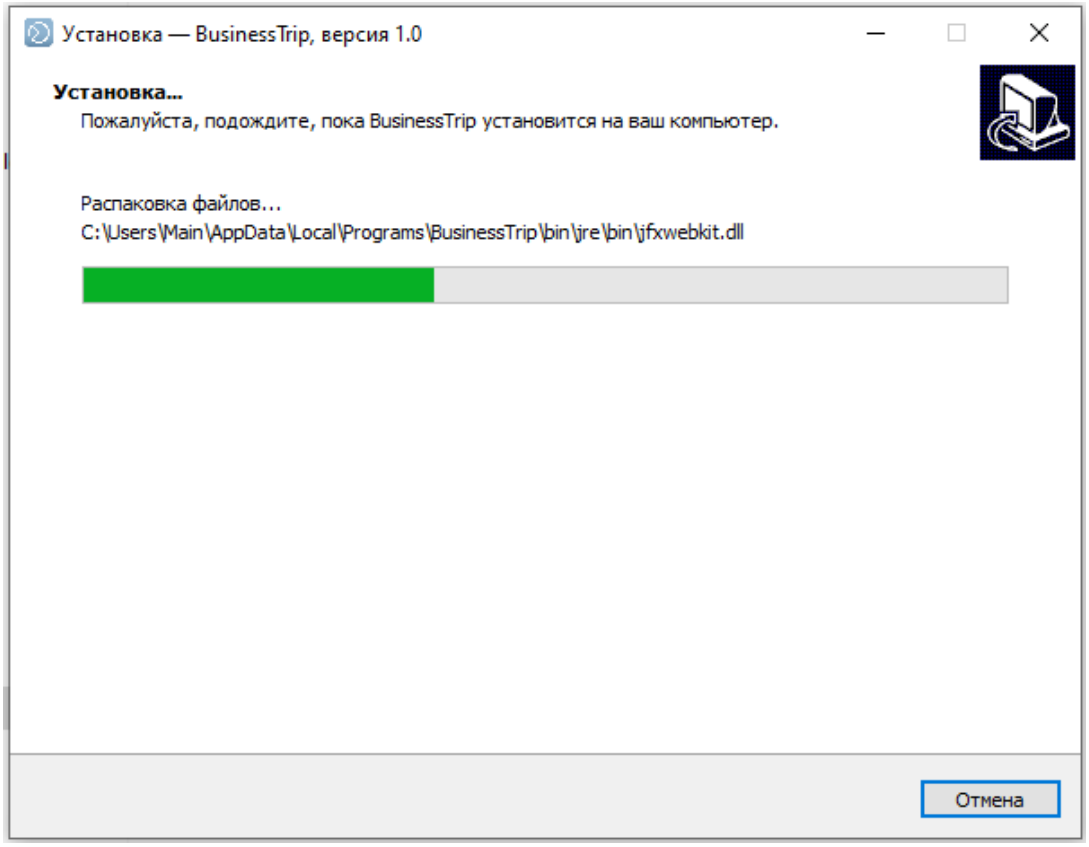

Рисунок 8.8 – Установка программы.

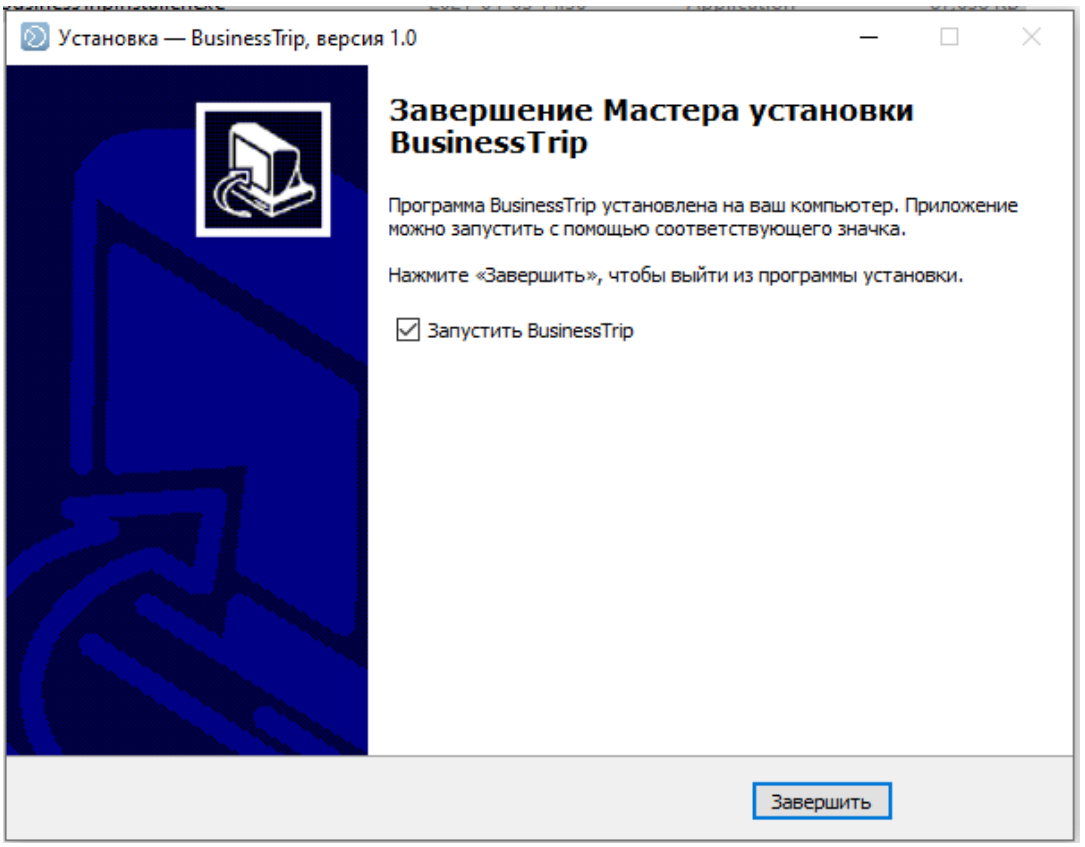

Рисунок 8.9 – Завершение установки программы.

8.3.3 Использование программы

Для начала работы с программой ее необходимо запустить. Если в процессе установки был создан ярлык на рабочем столе, то программу можно запустить двойным нажатием по ярлыку (Рисунок 8.10).

Рисунок 8.10 – Ярлык приложения BusinessTrip.

После запуска программы пользователь попадает на главный экран, главный экран изображен на рисунке 8.11. Только сотрудники, которые участвовали в какой-либо командировке отображаются в форме «Сотрудники».

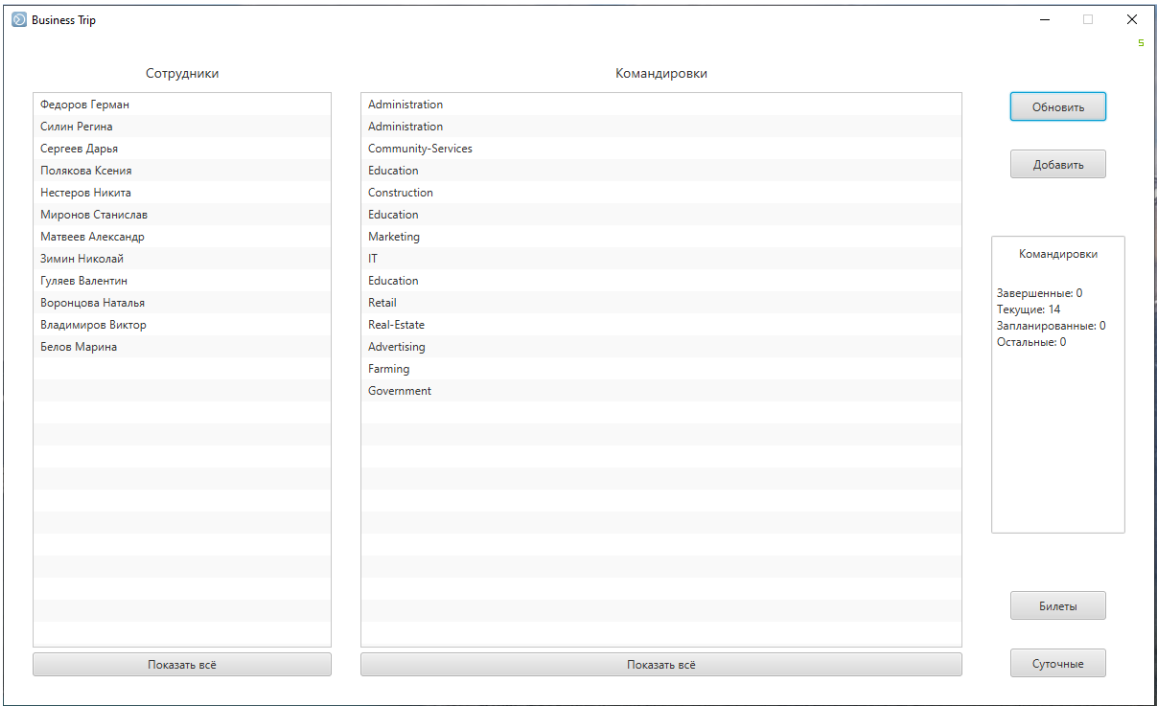

Рисунок 8.11 – Главный экран.

Для добавления сотрудника необходимо открыть список со всеми сотрудниками. Список сотрудников можно открыть по нажатию на кнопку «Показать всё». Список изображен на рисунке 8.12.

Далее необходимо нажать на кнопку добавить и заполнить всю информацию о сотруднике. Пример добавления сотрудника изображен на рисунке 8.13. Для сохранения информации о сотруднике нужно нажать на кнопку подтверждения «ОК». Окно с информацией о сотруднике автоматически закроется, а сотрудник будет добавлен в список.

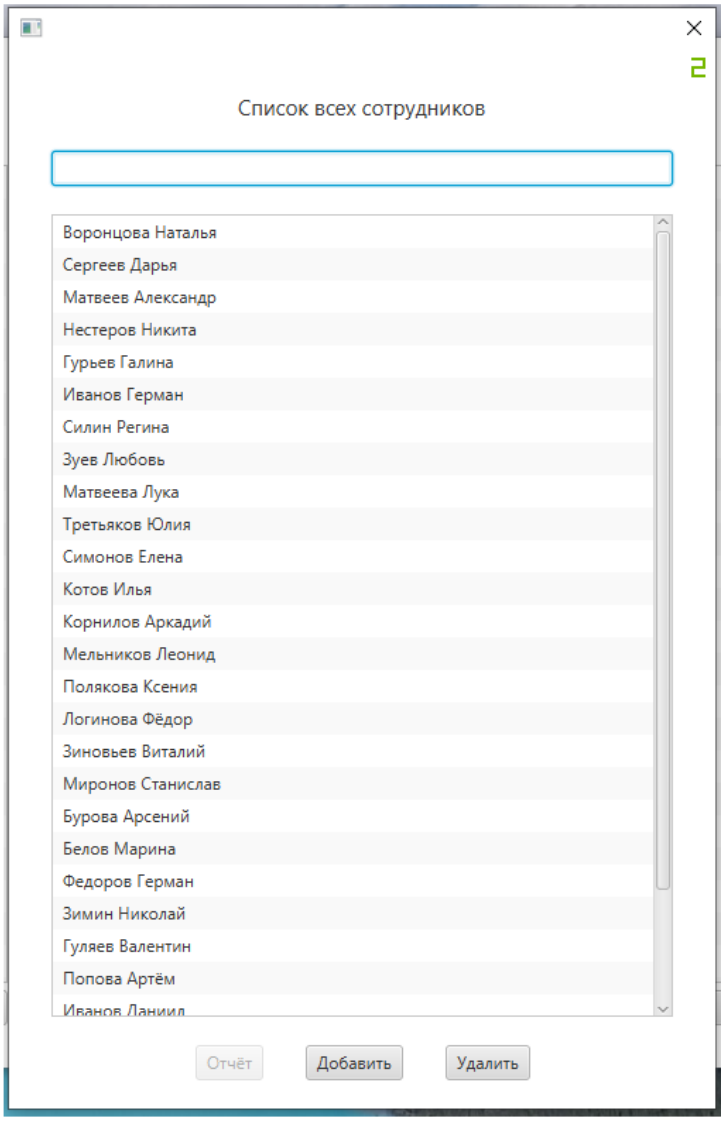

Рисунок 8.12 – Список сотрудников.

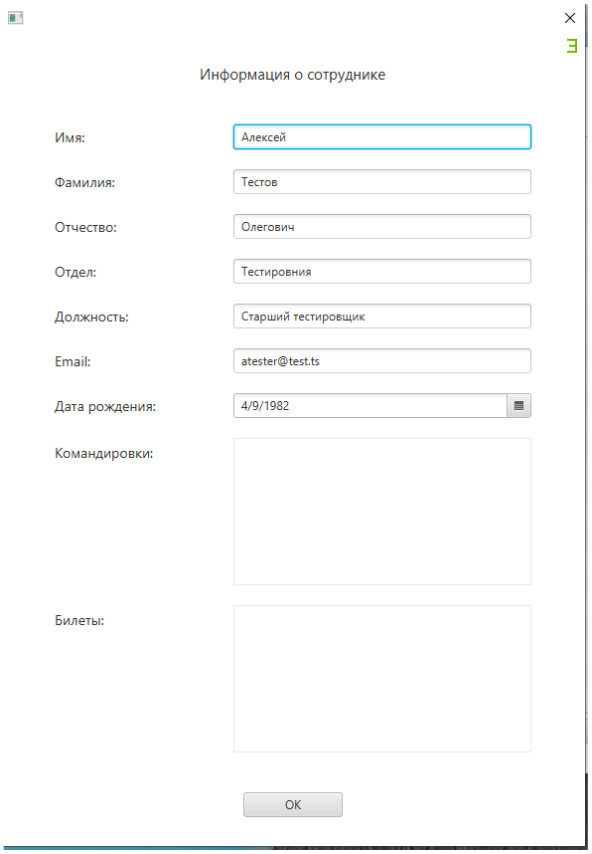

Рисунок 8.13 – Добавление сотрудника.

Для поиска сотрудника можно воспользоваться полем с поиском. Поиск будет производится по имени и фамилии сотрудника. Пример поиска сотрудника по фамилии изображен на рисунке 8.14.

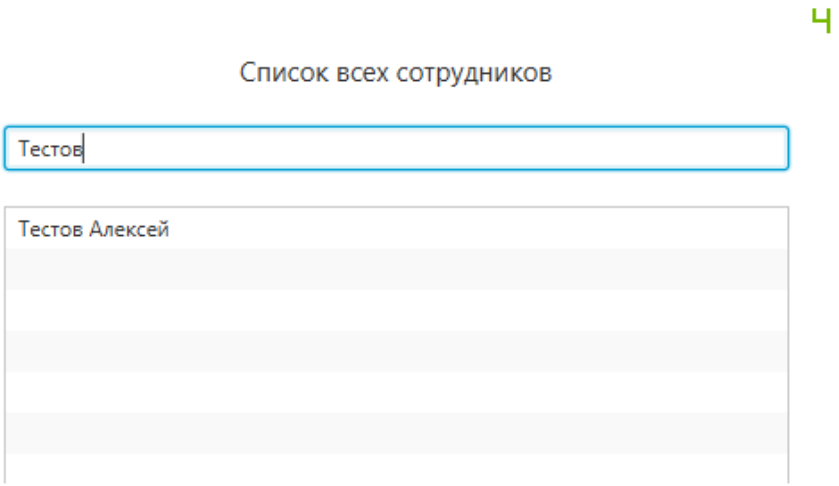

Рисунок 8.14 – Поиск сотрудников.

Чтобы изменить информацию о сотруднике необходимо 2 раза нажать на сотрудника, после откроется окно как на рисунке 8.13. На данном окне можно изменить любую информацию о сотруднике.

Таким образом был рассмотрен стандартный процесс добавления, поиска, изменения и удаления информации. Данный процесс аналогичен для билетов и суточных.

Для добавления командировки необходимо перейти на главное окно и нажать на кнопку Добавить. После нажатия откроется окно, изображенное на рисунке 8.15.

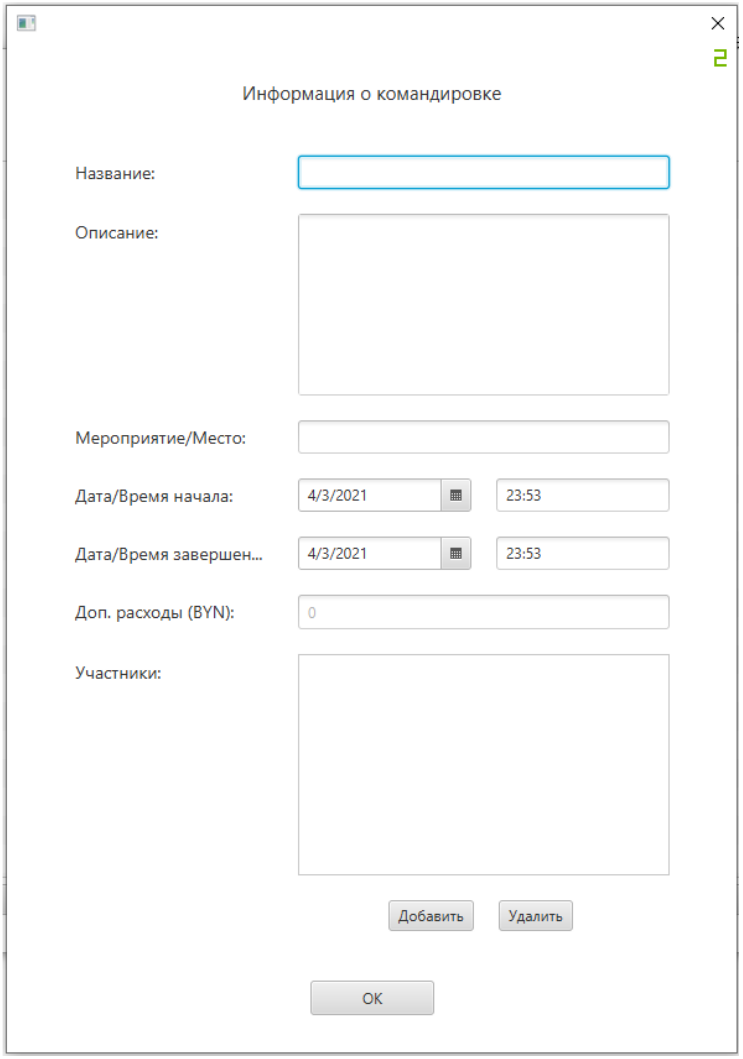

Рисунок 8.15 – Добавление командировки.

В командировку можно добавить несколько участников, для этого нужно нажать на кнопку Добавить в разделе Участники. После нажатия откроется окно, изображенное на рисунке 8.16. Можно выбрать любого сотрудника, а также добавить билеты и суточные. После заполнения всей информации необходимо нажать на кнопку ОК для сохранения информации.

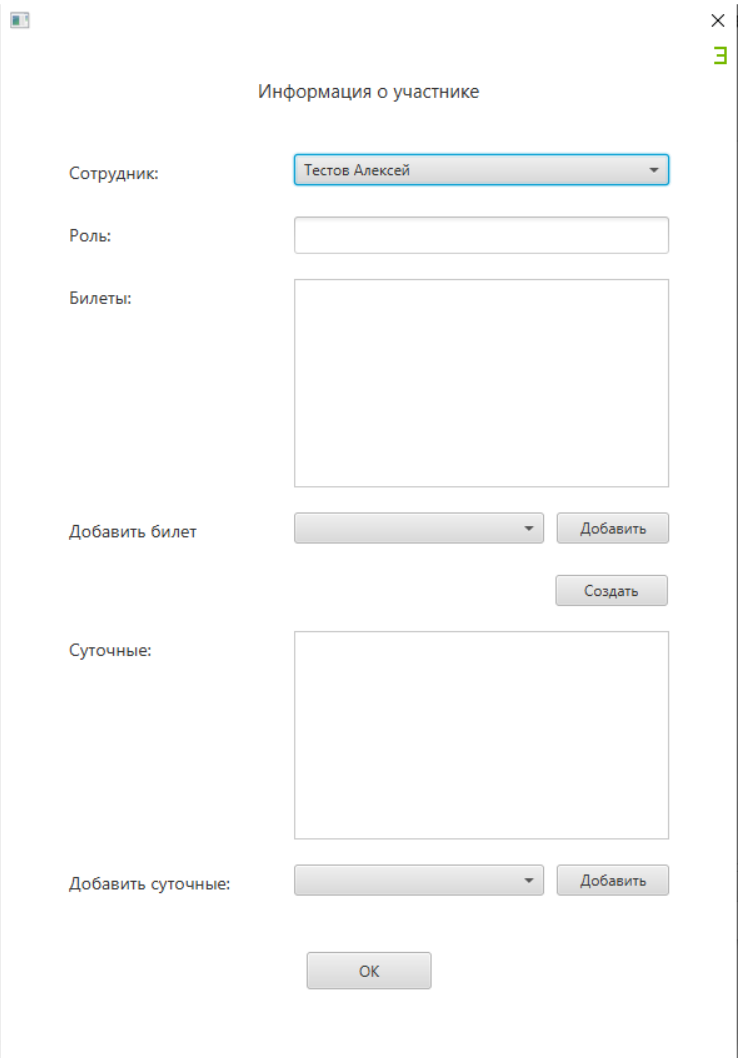

Рисунок 8.16 – Добавление участника.

Для создания отчета необходимо открыть список командировок. Окно со списком командировок изображено на рисунке 8.17. Далее нужно выбрать командировку и нажать на кнопку Отчет.

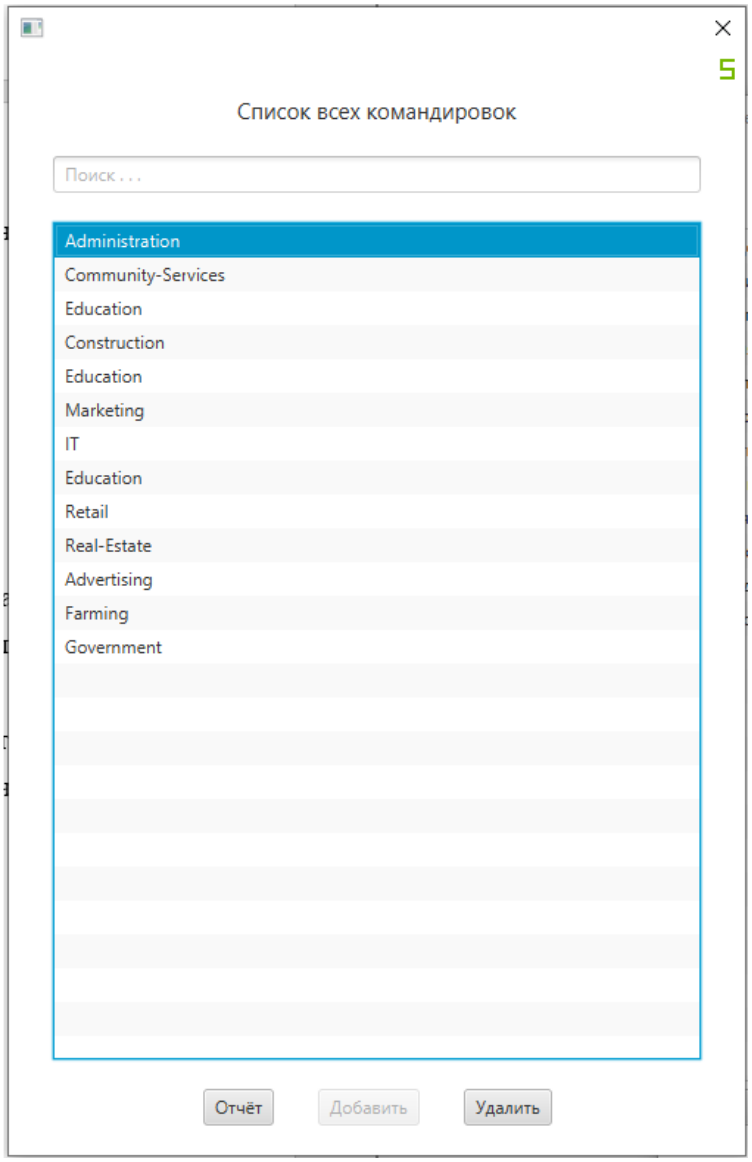

Рисунок 8.17 – Список командировок.

После нажатия отчет откроется в программе Microsoft Office Word, пример отчета изображен на рисунке 8.18. Отчеты будут сохранены в папке с программой.

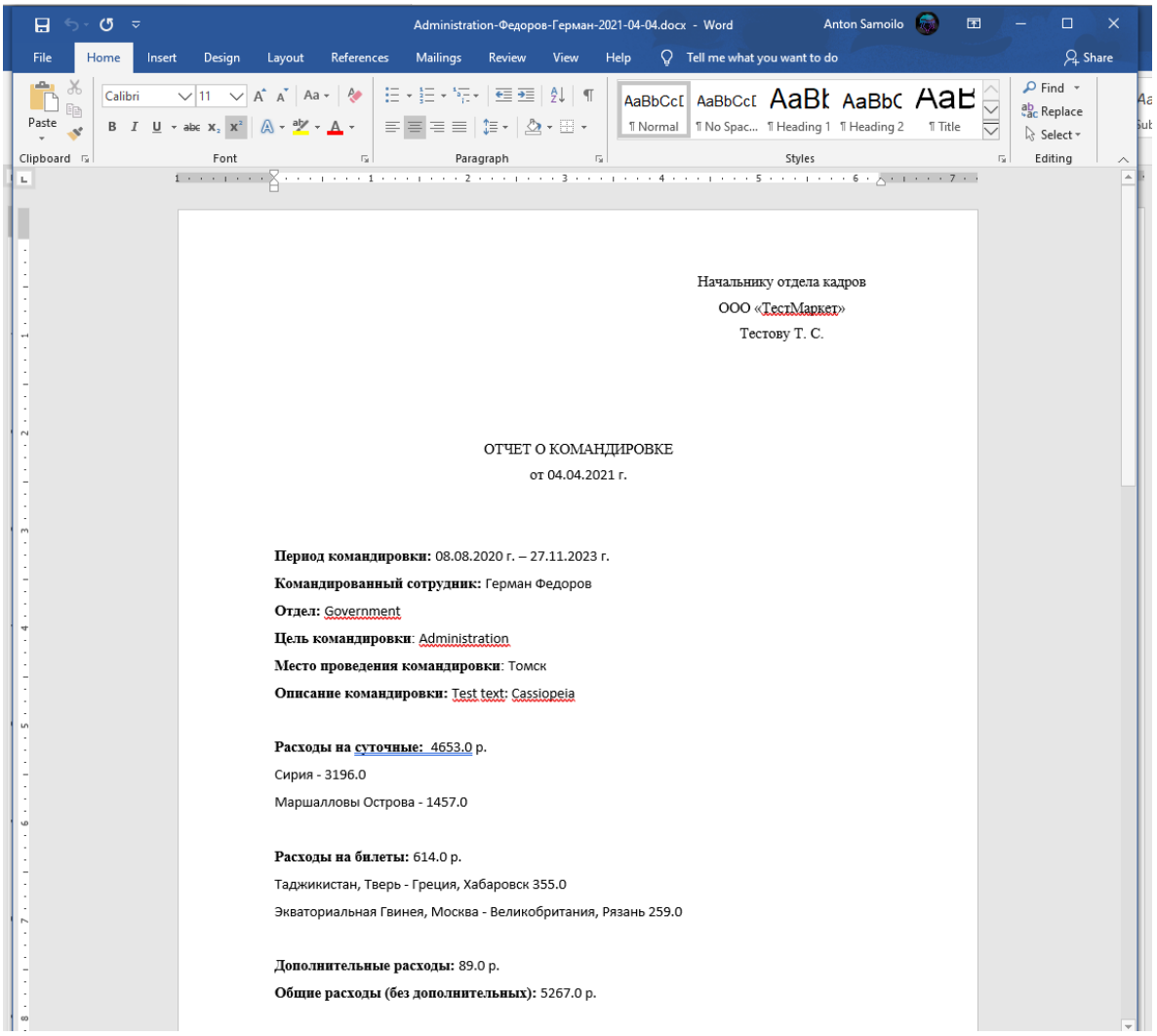

Рисунок 8.18 – Отчет о командировке.

В различных предприятиях могут использоваться разные виды отчетов, также отчеты могут меняться со временем. Для изменения шаблона отчета необходимо открыть папку с установленной программой. Далее открыть шаблон report.docx расположенный по пути \BusinessTrip\report\template.

Шаблон отчета – это простой документ, который содержит в себе ${ключи}, программа заменяет эти ключи на данные, так что можно использовать любой шаблон. Пример шаблона изображен на рисунке 8.19.

Начальнику отдела кадров

ООО «ТестМаркет»

Тестову Т. С.

ОТЧЕТ О КОМАНДИРОВКЕ

от ${date_now}

Период командировки: ${date_start} – ${date_end}

Командированный сотрудник: ${member}

Отдел: ${department}

Цель командировки: ${title}

Место проведения командировки: ${place}

Описание командировки: ${description}

Расходы на суточные: ${allowance_expenses}

${allowances}

Расходы на билеты: ${ticket_expenses}

${tickets}

Дополнительные расходы: ${additional_expenses}

Общие расходы (без дополнительных): ${expenses}

Рисунок 8.19 – Шаблон отчета.

8.4 Методы и средства защиты

Для упрощения работы, программа не имеет ограничений доступа. Любой, кто имеет доступ к компьютеру и программе сможет изменить данные, поэтому компьютер необходимо защитить паролем.

Для защиты информации в базе данных используется пароль. Пароль хранится в конфигурации программного продукта и в случае необходимости его можно будет изменить.

Заключение

Целью курсового проекта являлось создание программы позволяющей автоматизировать процесс создания и редактирования информации о командировках. Особенностью данной программы является то, что любую информацию можно поменять в любой момент, а также все запросы на изменение данных выполняются как транзакции. Был использован read committed уровень изолированности транзакций, данный уровень имеет высокую скорость выполнения и среднюю согласованность данных, предоставляя защиту от чтения незафиксированных изменений другой транзакцией.

В процессе выполнения курсового проекта была подробно изучена деятельность предприятия, а также обязанности секретаря, что позволило составить подробные требования к программному продукту. Перед началом разработки были проанализированы схожие по функционалу программы и были выявлены их достоинства и недостатки.

В процессе разработки программного продукта была спроектирована база данных, которая позволила хранить всю необходимую информацию. Также в случае необходимости есть возможность заменить базу данных на любую другую, что и было сделано. Из-за сложности с настройкой PostgreSQL было принято решение добавить поддержку H2 Database. Для работы H2 не нужны какие-либо дополнительные программы.

Для создания программы был использован язык программирования java, с помощью которого были реализованы все необходимые функции.

Также архитектура, используемая в данном приложении, позволит легко и просто расширять функционал в любых направлениях.

После завершения разработки, программа была полностью протестирована. Также для конечных пользователей разработано подробное руководство по установке и использованию приложения.

Все задачи были выполнены в полном объёме. Была подробно изучена деятельность секретаря и предприятия, на основе полученных данных был подготовлен набор требований к программному обеспечению, а также были проанализированы аналоги.

Цель курсового проекта достигнута, было разработано приложение позволяющее автоматизировать процесс создания и редактирования информации о командировках. Разработанное программное обеспечение соответствует требованиям, и может быть использовано конечными пользователями.

Список использованных источников

[1] Диаграммы UML [Электронный ресурс]. – Режим доступа: https://www.visual-paradigm.com/guide/ – Дата доступа: 10.03.2021.

[2] Модель данных [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Model_dannich – Дата доступа: 10.03.2021.

[3] Microsoft [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/ru-ru/windows – Дата доступа: 10.03.2021.

[4] Jetbrains [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/idea/ – Дата доступа: 10.03.2021.

[5] Java документация [Электронный ресурс]. – Режим доступа: https://docs.oracle.com/en/java/ – Дата доступа: 10.03.2021.

[6] Справочник Java [Электронный ресурс]. – Режим доступа: https://mkyong.com/ – Дата доступа: 10.03.2021.

[7] Руководство Java [Электронный ресурс]. – Режим доступа: https://www.baeldung.com/ – Дата доступа: 10.03.2021.

[8] Руководство по Spring Boot [Электронный ресурс]. – Режим доступа: https://spring.io/guides/ – Дата доступа: 10.03.2021.

[9] Документация по postgresql [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/ – Дата доступа: 10.03.2021.

[10] Документация по H2 [Электронный ресурс]. – Режим доступа: https://www.h2database.com/html/main.html – Дата доступа: 10.03.2021.

[11] Junit 5 [Электронный ресурс]. – Режим доступа: https://junit.org/junit5/ – Дата доступа: 10.03.2021.

Приложение А

(обязательное)

Текст программы

```
@SpringBootApplication(scanBasePackages = "com.qualle.trip")
@ComponentScan("com.qualle.trip")
@EnableJpaRepositories("com.qualle.trip.repository")
@EntityScan("com.qualle.trip.model.entity")
public class Application extends AbstractJavaFxSupport {

    @Autowired
    private ViewHolder mainView;

    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("Business Trip");
        stage.getIcons().add(new Image("/icon.png"));
        stage.setScene(new Scene(mainView.getView()));
        stage.setResizable(false);
        stage.centerOnScreen();
        stage.show();
    }

    public static void main(String[] args) {
        launchApp(Application.class, args);
    }
}


@Service
public class AllowanceServiceImpl implements AllowanceService {

    @Autowired
    private MemberAllowanceDao memberAllowanceDao;
```

```
@Autowired
private AllowanceDao allowanceDao;

@Autowired
private MemberService memberService;

@Override
public List<Allowance> getAll() {
   return allowanceDao.getAll();
}

@Override
public List<AllowanceDto> getAllDto() {
   return toDtoArray(getAll());
}

@Override
public List<MemberAllowanceDto> getAllAllowanceDtoByMember(long memberId) {
   return toMemberDtoArray(memberAllowanceDao.getAllByMember(memberId));
}

@Override
public Allowance getById(long id) {
   return allowanceDao.getById(id);
}

@Override
public AllowanceDto getDtoById(long id) {
   return toDto(getById(id));
}

@Override
public List<AllowanceDto> getDtoByCountry(String country) {
   return toDtoArray(allowanceDao.getByCountry(country));
}

@Override
@Transactional
public void add(MemberAllowanceDto dto) {
   Allowance allowance = allowanceDao.getById(dto.getAllowance().getId());
   Member member = memberService.getById(dto.getMember().getId());
   MemberAllowance memberAllowance = new MemberAllowance(allowance, member, dto.getDays());
   memberAllowanceDao.add(memberAllowance);
}

@Override
@Transactional
public void add(AllowanceDto dto) {
   allowanceDao.add(new Allowance(dto.getValue(), dto.getCountry()));
}

@Override
@Transactional
public void update(AllowanceDto dto) {
   Allowance allowance = getById(dto.getId());
   allowance.setCountry(dto.getCountry());
   allowance.setValue(dto.getValue());
   allowanceDao.update(allowance);
```

```
    }

    @Override
    @Transactional
    public void delete(long allowanceId, long memberId) {
      memberAllowanceDao.delete(allowanceId, memberId);
    }

    @Override
    @Transactional
    public void delete(long id) {
      allowanceDao.delete(id);
    }

    @Override
    public AllowanceDto toDto(Allowance allowance) {
      AllowanceDto dto = new AllowanceDto(allowance.getCountry(), allowance.getValue());
      dto.setId(allowance.getId());
      return dto;
    }

    @Override
    public MemberAllowanceDto toMemberDto(MemberAllowance memberAllowance) {
      return new MemberAllowanceDto(toDto(memberAllowance.getAllowance()),
memberAllowance.getDays());
    }

    @Override
    public List<AllowanceDto> toDtoArray(Collection<Allowance> allowances) {
      List<AllowanceDto> dto = new ArrayList<>();
      for (Allowance allowance : allowances) {
        dto.add(toDto(allowance));
      }
      return dto;
    }

    @Override
    public List<MemberAllowanceDto> toMemberDtoArray(Collection<MemberAllowance>
memberAllowances) {
      List<MemberAllowanceDto> dto = new ArrayList<>();
      for (MemberAllowance memberAllowance : memberAllowances) {
        dto.add(toMemberDto(memberAllowance));
      }
      return dto;
    }
  }


  @Service
  public class EmployeeServiceImpl implements EmployeeService {

    @Autowired
    private EmployeeDao employeeDao;

    @Autowired
    private TripService tripService;

    @Autowired
```

```
private TicketService ticketService;

@Override
public List<Employee> getAll() {
   return employeeDao.getAll();
}

@Override
public List<EmployeeDto> getAllDto() {
   return toDtoArray(getAll());
}

@Override
public List<EmployeeSimpleDto> getAllSimpleDto() {
   return toSimpleDtoArray(getAll());
}

@Override
public List<EmployeeSimpleDto> getAllSimpleDtoByTrip() {
   return toSimpleDtoArray(employeeDao.getByTrip());
}

@Override
public List<EmployeeSimpleDto> getAllSimpleDtoByTrip(long tripId) {
   return toSimpleDtoArray(employeeDao.getByTrip(tripId));
}

@Override
public Employee getById(long id) {
   return employeeDao.getById(id);
}

@Override
public EmployeeDto getDtoById(long id) {
   return toDto(getById(id));
}

@Override
public EmployeeDto getFullDtoById(long id) {
   Employee employee = employeeDao.getFullById(id);
   EmployeeDto dto = toDto(employee);
   for (Member member : employee.getMembers()) {
      if (dto.getTickets() == null && dto.getTrips() == null) {
         dto.setTickets(new ArrayList<>());
         dto.setTrips(new ArrayList<>());
      }
      dto.getTickets().addAll(ticketService.toDtoArray(member.getTickets()));
      dto.getTrips().add(tripService.toSimpleDto(member.getTrip()));
   }
   return dto;
}

@Override
public List<EmployeeSimpleDto> getSimpleDtoByName(String name) {
   return toSimpleDtoArray(employeeDao.getByName(name));
}

@Override
```

```
    @Transactional
    public void add(EmployeeDto dto) {
        Employee employee = new Employee(dto.getName(), dto.getSurname(), dto.getPatronymic(),
dto.getPosition(), dto.getDepartment(), dto.getEmail(), dto.getBirthday());
        employeeDao.add(employee);
    }

    @Override
    @Transactional
    public void update(EmployeeDto dto) {
        Employee employee = getById(dto.getId());
        employee.setName(dto.getName());
        employee.setSurname(dto.getSurname());
        employee.setEmail(dto.getEmail());
        employee.setDepartment(dto.getDepartment());
        employee.setBirthday(dto.getBirthday());
        employeeDao.update(employee);
    }

    @Override
    @Transactional
    public void delete(long id) {
        employeeDao.delete(id);
    }

    @Override
    public EmployeeDto toDto(Employee employee) {
        EmployeeDto dto = new EmployeeDto(employee.getName(), employee.getSurname(),
employee.getPatronymic(), employee.getPosition(), employee.getDepartment(), employee.getEmail(),
employee.getBirthday());
        dto.setId(employee.getId());
        return dto;
    }

    @Override
    public EmployeeSimpleDto toSimpleDto(Employee employee) {
        EmployeeSimpleDto dto = new EmployeeSimpleDto(employee.getName(), employee.getSurname(),
employee.getEmail());
        dto.setId(employee.getId());
        return dto;
    }

    @Override
    public List<EmployeeDto> toDtoArray(Collection<Employee> employees) {
        List<EmployeeDto> dto = new ArrayList<>();
        for (Employee employee : employees) {
            dto.add(toDto(employee));
        }
        return dto;
    }

    @Override
    public List<EmployeeSimpleDto> toSimpleDtoArray(Collection<Employee> employees) {
        List<EmployeeSimpleDto> dto = new ArrayList<>();
        for (Employee employee : employees) {
            dto.add(toSimpleDto(employee));
        }
        return dto;
```

```
    }
}

@Service
public class MemberServiceImpl implements MemberService {

    @Autowired
    private MemberDao memberDao;

    @Autowired
    private EmployeeService employeeService;

    @Autowired
    private TripService tripService;

    @Autowired
    private TicketService ticketService;

    @Autowired
    private AllowanceService allowanceService;

    @Override
    public List<MemberSimpleDto> getAllDto() {
        return toSimpleDtoArray(memberDao.getAll());
    }

    @Override
    public Member getById(long id) {
        return memberDao.getById(id);
    }

    @Override
    public MemberDto getDtoById(long id) {
        return toDto(getById(id));
    }

    @Override
    @Transactional
    public void add(MemberDto dto) {
        Member member = new Member();
        member.setEmployee(employeeService.getById(dto.getEmployee().getId()));
        member.setTrip(tripService.getById(dto.getTrip().getId()));
        member.setRole(dto.getRole());
        memberDao.add(member);
    }

    @Override
    @Transactional
    public void update(MemberDto dto) {
        Member member = memberDao.getById(dto.getId());
        member.setRole(dto.getRole());
        memberDao.update(member);
    }

    @Override
    @Transactional
    public void delete(long id) {
        memberDao.delete(id);
```

```
        }

        @Override
        public MemberDto toDto(Member member) {
            double ticketsExpenses = member.getTickets().stream().mapToDouble(Ticket::getPrice).sum();
            double allowancesExpenses = member.getMemberAllowances().stream().mapToDouble(m ->
m.getDays() * m.getAllowance().getValue()).sum();
            MemberDto dto = new MemberDto(allowancesExpenses, ticketsExpenses,
employeeService.toDto(member.getEmployee()), tripService.toDto(member.getTrip()),
                    allowanceService.toMemberDtoArray(member.getMemberAllowances()),
ticketService.toDtoArray(member.getTickets()));
            dto.setId(member.getId());
            return dto;
        }

        @Override
        public MemberSimpleDto toSimpleDto(Member member) {
            MemberSimpleDto dto = new MemberSimpleDto(member.getId(), member.getEmployee().getName(),
member.getEmployee().getSurname(), member.getRole());
            dto.setId(member.getId());
            return dto;
        }

        @Override
        public List<MemberSimpleDto> toSimpleDtoArray(Collection<Member> members) {
            List<MemberSimpleDto> dto = new ArrayList<>();
            for (Member member : members) {
                dto.add(toSimpleDto(member));
            }
            return dto;
        }
    }

    @Service
    public class TicketServiceImpl implements TicketService {

        @Autowired
        private TicketDao ticketDao;

        @Autowired
        private EmployeeService employeeService;

        @Override
        public List<Ticket> getAll() {
            return ticketDao.getAll();
        }

        @Override
        public List<TicketDto> getAllDto() {
            return toDtoArray(getAll());
        }

        @Override
        public List<TicketDto> getAllDtoWithoutMember() {
            return toDtoArray(ticketDao.getAllWithoutMember());
        }

        @Override
```

```
public List<TicketDto> getDtoByMember(long memberId) {
   return toDtoArray(ticketDao.getByMember(memberId));
}

@Override
public List<TicketDto> getDtoByEmployeeAndTrip(long employeeId, long tripId) {
   return toDtoArray(ticketDao.getByEmployeeAndTrip(employeeId, tripId));
}

@Override
public Ticket getById(long id) {
   return ticketDao.getById(id);
}

@Override
public TicketDto getDtoById(long id) {
   return toDto(getById(id));
}

@Override
public TicketDto getFullDtoById(long id) {
   Ticket ticket = ticketDao.getFullById(id);
   TicketDto dto = toDto(ticket);
   if (ticket.getMember() != null) {
      dto.setEmployee(employeeService.toSimpleDto(ticket.getMember().getEmployee()));
   }
   return dto;
}

@Override
public List<TicketDto> getDtoByLocation(String location) {
   return toDtoArray(ticketDao.getByLocation(location));
}

@Override
@Transactional
public void add(TicketDto dto) {
   Ticket ticket = new Ticket(dto.getFrom(), dto.getTo(), new Date(), dto.getPrice(), dto.getType());
   ticketDao.add(ticket);
}

@Override
@Transactional
public void update(TicketDto dto) {
   Ticket ticket = getById(dto.getId());
   ticket.setFrom(dto.getFrom());
   ticket.setTo(dto.getTo());
   ticket.setDate(dto.getDate());
   ticket.setPrice(dto.getPrice());
   ticket.setType(dto.getType());
   ticketDao.update(ticket);
}

@Override
@Transactional
public void delete(long id) {
   ticketDao.delete(id);
}
```

```
    @Override
    public TicketDto toDto(Ticket ticket) {
        TicketDto dto = new TicketDto(ticket.getFrom(), ticket.getTo(), ticket.getDate(), ticket.getPrice(),
ticket.getType());
        dto.setId(ticket.getId());
        return dto;
    }

    @Override
    public List<TicketDto> toDtoArray(Collection<Ticket> tickets) {
        List<TicketDto> dto = new ArrayList<>();
        for (Ticket ticket : tickets) {
            dto.add(toDto(ticket));
        }
        return dto;
    }
}


@Service
@RequiredArgsConstructor
public class TripServiceImpl implements TripService {

    @Autowired
    private TripDao tripDao;

    @Autowired
    private MemberService memberService;

    @Autowired
    private EmployeeService employeeService;

    @Autowired
    private AllowanceService allowanceService;

    @Autowired
    private TicketService ticketService;

    @Override
    public List<Trip> getAll() {
        return tripDao.getAll();
    }

    @Override
    public List<TripSimpleDto> getAllSimpleDto() {
        return toSimpleDtoArray(getAll());
    }

    @Override
    public List<TripSimpleDto> getAllSimpleDtoByEmployee(long employeeId) {
        return toSimpleDtoArray(tripDao.getByEmployee(employeeId));
    }

    @Override
    public Trip getById(long id) {
        return tripDao.getById(id);
    }
```

```
    @Override
    public TripDto getDtoById(long id) {
        return toDto(getById(id));
    }

    @Override
    public TripSimpleDto getSimpleDtoById(long id) {
        return toSimpleDto(getById(id));
    }

    @Override
    public List<TripSimpleDto> getSimpleDtoByTitle(String title) {
        return toSimpleDtoArray(tripDao.getByTitle(title));
    }

    @Override
    public TripDto getFullDtoById(long id) {
        Trip trip = tripDao.getFullById(id);
        TripDto dto = toDto(trip);

        dto.setStatus(getStatus(trip.getStart(), trip.getEnd()));
        dto.setTicketExpenses(ExpensesCalculator.calcTicketExpenses(trip.getMembers()));
        dto.setAllowanceExpenses(ExpensesCalculator.calcAllowanceExpenses(trip.getMembers()));
        dto.setExpenses(dto.getTicketExpenses() + dto.getAllowanceExpenses() +
trip.getAdditionalExpenses());

        dto.setMembers(trip.getMembers().stream().map(m ->
memberService.getDtoById(m.getId())).collect(Collectors.toList()));

        return dto;
    }

    @Override
    @Transactional
    public void add(TripDto dto) {
        Trip trip = new Trip(dto.getTitle(), dto.getDescription(), dto.getPlace(), dto.getAdditionalExpenses());
        trip.setStart(dto.getStart());
        trip.setEnd(dto.getEnd());
        trip.setMembers(new HashSet<>());

        for (MemberDto memberDto : dto.getMembers()) {
            Member member = new Member();
            member.setTrip(trip);
            member.setRole(memberDto.getRole());
            member.setEmployee(employeeService.getById(memberDto.getEmployee().getId()));
            member.setTickets(memberDto.getTickets().stream().map(t ->
ticketService.getById(t.getId())).collect(Collectors.toSet()));
            member.getTickets().forEach(t -> t.setMember(member));
            member.setMemberAllowances(memberDto.getAllowances().stream().map(a -> new
MemberAllowance(allowanceService.getById(a.getAllowance().getId()), member,
a.getDays())).collect(Collectors.toSet()));
            trip.getMembers().add(member);
        }
        tripDao.add(trip);
    }

    @Override
```

```
    @Transactional
    public void update(TripDto dto) {
        Trip trip = tripDao.getById(dto.getId());
        trip.setTitle(dto.getTitle());
        trip.setDescription(dto.getDescription());
        trip.setPlace(dto.getPlace());
        trip.setAdditionalExpenses(dto.getAdditionalExpenses());
        tripDao.update(trip);
    }

    @Override
    @Transactional
    public void delete(long id) {
        tripDao.delete(id);
    }

    @Override
    public TripDto toDto(Trip trip) {
        TripDto dto = new TripDto(trip.getTitle(), trip.getDescription(), trip.getPlace(), trip.getStart(),
trip.getEnd(), trip.getAdditionalExpenses());
        dto.setId(trip.getId());
        return dto;
    }

    @Override
    public TripSimpleDto toSimpleDto(Trip trip) {
        TripSimpleDto dto = new TripSimpleDto(trip.getTitle(), trip.getDescription(),
trip.getAdditionalExpenses(), getStatus(trip.getStart(), trip.getEnd()));
        dto.setId(trip.getId());
        return dto;
    }

    @Override
    public List<TripSimpleDto> toSimpleDtoArray(Collection<Trip> trips) {
        List<TripSimpleDto> dto = new ArrayList<>();
        for (Trip trip : trips) {
            dto.add(toSimpleDto(trip));
        }
        return dto;
    }

    private static TripStatus getStatus(Date start, Date end) {
        Date now = new Date();

        if (now.before(start) && now.before(end)) {
            return TripStatus.FUTURE;
        } else if (now.after(start) && now.before(end)) {
            return TripStatus.IN_PROGRESS;
        } else {
            return TripStatus.COMPLETED;
        }
    }
}

@Slf4j
@Component
@RequiredArgsConstructor
public class WordReportService implements ReportService {
```

```
private static final String DELIMITER = "-";

private final ReportConfig reportConfig;

@Override
public void make(TripDto trip) {
   for (MemberDto member : trip.getMembers()) {
      createMemberReport(trip, member);
   }
}

private void createMemberReport(TripDto trip, MemberDto member){
   String targetPath = reportConfig.getOutputPath() + getFileName(trip, member);

   WordUtil.createReport(targetPath, reportConfig.getTemplatePath(), getData(trip, member));
}

private String getFileName(TripDto trip, MemberDto member) {
   StringBuilder name = new StringBuilder();

   name.append(trip.getTitle())
         .append(DELIMITER)
         .append(member.getEmployee().getSurname())
         .append(DELIMITER)
         .append(member.getEmployee().getName())
         .append(DELIMITER)
         .append(getCurrentDate())
         .append(".docx");

   return name.toString();
}

private Map<String, Object> getData(TripDto trip, MemberDto member) {
   Map<String, Object> data = new HashMap<>();
   data.put("date_now", formatDate(new Date()));
   data.put("date_start", formatDate(trip.getStart()));
   data.put("date_end", formatDate(trip.getEnd()));
   data.put("title", trip.getTitle());
   data.put("place", trip.getPlace());
   data.put("description", trip.getDescription());
   data.put("additional_expenses", trip.getAdditionalExpenses() + " p.");
   fillMemberData(data, member);
   return data;
}

private void fillMemberData(Map<String, Object> data, MemberDto member) {
   data.put("member", member.getEmployee().getName() + " " + member.getEmployee().getSurname());
   data.put("department", member.getEmployee().getDepartment());
   data.put("allowance_expenses", member.getAllowanceExpenses() + " p.");
   data.put("allowances", getAllowanceInfo(member.getAllowances()));
   data.put("ticket_expenses", member.getTicketsExpenses() + " p.");
   data.put("tickets", getTicketInfo(member.getTickets()));
   data.put("expenses", member.getTicketsExpenses() + member.getAllowanceExpenses() + " p.");
}

private String getCurrentDate() {
   return new SimpleDateFormat("yyyy-MM-dd").format(new Date());
```

```
    }
}
```