**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 2](#_Toc135725423)

[ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc135725424)

[1.1 Основные понятия 4](#_Toc135725425)

[1.2 Функционал информационной системы 5](#_Toc135725426)

[1.3 Используемые программы и инструменты 6](#_Toc135725427)

[1.3.1 Средства проектирования 6](#_Toc135725428)

[1.3.2 Серверная часть 12](#_Toc135725429)

[1.3.3 Клиентская часть 13](#_Toc135725430)

[1.3.4 Среда программирования 13](#_Toc135725431)

[1.3.5 Иные вспомогательные программные средства 15](#_Toc135725432)

# ВВЕДЕНИЕ

Процессы, проходящие на железнодорожных вокзалах, обширны. Обслуживание большого пассажиропотока, формирование соответствующих бизнес процессов, работа с расписанием и контроль состояния облачных кошельков – все это относится к спектру обязанностей персонала железнодорожных станций.

Зачастую человеческий фактор вносит существенную лепту в ход работы подобных процессов. Нерадивый кассир своей нерасторопностью может остановить поток пассажиров и создать спонтанную очередь. Неправильно составленное расписание может привести к неправильному восприятию пассажирами маршрута и негативным отзывам в дальнейшем.

Подавляющее большинство такого рода ситуаций можно предотвратить путем создания автоматизированной системы, позволяющей оптимизировать работу диспетчера железнодорожной кассы и свести человеческий фактор к минимуму.

Проектирование и реализация системы, позволяющей автоматизировать работу диспетчера железнодорожной кассы, является целью данной работы.

Объект исследования: деятельность ООО «Гринривер».

Предмет исследования: автоматизация деятельности диспетчера железнодорожной кассы по продаже билетов

В соответствии с целью поставлены следующие задачи исследования:

* Проанализировать деятельность ООО «Гринривер» и выявить информационные потоки;
* Провести сравнительный анализ аналогов разрабатываемой информационной системы.
* Обосновать выбор инструментальных средств и СУБД для разработки информационной системы.
* Провести структурное (функциональное) и объектно-ориентированное моделирование информационной системы.
* Спроектировать базу данных
* Протестировать информационную систему и разработать документацию для диспетчера, администратора системы.

Структура ВКР включает в себя содержание работы, введение, три главы, заключение, список использованных источников.

Во введении обоснована актуальность темы, определены цели, задачи, предмет и объект исследования.

Первая глава посвящена анализу теоретической информации по заявленной теме исследования, формированию перечня функций, выбору обеспечения для разработки.

Вторая глава содержит описание проектирования информационной системы, дизайн интерфейса, практическую реализацию с помощью моделей DFD, IDEF0, ERD.

Третья глава практическая. Создание пользовательского интерфейса, реализация основных программных модулей, компиляцию и тестирование программы.

# 1. Исследование процесса работы железнодорожного вокзала

## 1.1 Анализ предметной области

## 1.2 Обзор аналогов проектируемой системы

## 1.3 Исследование информационных потоков ИС

## 1.4 Формальная постановка задач для ИС

Основным направлением деятельности ООО «Гринривер» является разработка информационных систем.

От компании, занимающейся железнодорожным обслуживанием, поступил заказ на разработку данной информационной системы. Внедрение программы позволит ускорить работу с пассажиропотоком, что снизит очереди и вероятность непредвиденных ситуаций. Таким образом, разработку программы можно разбить на следующие этапы:

1. Получение технического задания от заказчика с четким описанием того, что именно должно быть реализовано в разрабатываемом ПО.
2. Проектирование диаграмм всех указанных процессов. Наиболее практичными являются методологии DFD, IDEF0, ERD.
3. Следующий шаг – это определить роли системы. Реализовуемая программа представляет из себя двухфазную информационную систему, что подразумевает реализацию серверной части. Следовательно, если у пользователя имеется доступ к определенному набору данных в базе данных, регулирование доступа реализуется за счет системы ролей. Иной системы распределения доступа компании не требуется.
4. Последний этап проектирования (с точки зрения заказчика) – это ментальная проверка как все это будет работать.
5. Начало реализации программной части работы, создание пользовательского интерфейса, создание программных модулей, формирование основных алгоритмических структур.
6. Завершение создания автоматизированной системы.
7. Тестирование.

Разрабатываемая система должна выполнять следующие функции:

* Создание, редактирование и удаление рейсов;
* Создание новых пользователей, авторизация;
* Продажа диспетчером билетов;
* Покупка, управление собственными билетами для пользователей;
* Возможность добавлять и редактировать диспетчеров;
* Возможность добавлять новые сервисы для пополнения электронного кошелька;
* Наличие точек выхода-заглушек для реализации электронных кассовых систем;

## 1.3 Используемые программы и инструменты

Реализация любой информационной системы обязательно начинается с выбора необходимых для её реализации средств, в частности средств проектирования, языка программирования и любого дополнительного программного обеспечения.

### 1.3.1 Средства проектирования

Для проектирования автоматизированной системы управления цифровыми двойниками предприятия использовалась среда проектирования диаграмм ER Ramus, а также программа визуализации графических диаграмм Draw.Io.

Ramus - это программа формирования визуальных диаграмм, которая используется для графического моделирования различных бизнес-процессов. Помимо визуализации процессов и задач, создаваемые диаграммы также подходят для классификации и организации данных. Основное преимущество программы в том, что она поддерживает одновременно две популярные методологии: DFD и IDEF0.

DFD **-** методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ.

Непосредственно DFD нотация состоит из следующих элементов (рисунок 1.4):

* Процесс (англ. Process), т.е. функция или последовательность действий, которые нужно предпринять, чтобы данные были обработаны. Это может быть создание цифрового двойника, создание компонента, сохранение и т.д. В названиях процессов принято использовать глаголы, т.е. «Создать компонент» (а не «создание компонента») или «сохранить двойника» (а не «сохранение двойника»). Здесь нет строгой системы требований, как, например, в IDEF0 или BPMN, где нотации имеют жестко определенный синтаксис, так как они могут быть исполняемыми. Но все же определенных правил стоит придерживаться, чтобы не вносить путаницу при чтении DFD другими людьми;
* Внешние сущности (англ. External Entity). Это любые объекты, которые не входят в саму систему, но являются для нее источником информации либо получателями какой-либо информации из системы после обработки данных. Это может быть человек, внешняя система, какие-либо носители информации и хранилища данных;
* Хранилище данных (англ. Data store). Внутреннее хранилище данных для процессов в системе. Поступившие данные перед обработкой и результат после обработки, а также промежуточные значения должны где-то храниться. Это и есть базы данных, таблицы или любой другой вариант организации и хранения данных. Здесь будут храниться данные о клиентах, заявки клиентов, расходные накладные и любые другие данные, которые поступили в систему или являются результатом обработки процессов;
* Поток данных (англ. Data flow). В нотации отображается в виде стрелок, которые показывают, какая информация входит, а какая исходит из того или иного блока на диаграмме.

Нотация DFD может описывать любые действия, в том числе, процесс продажи или отгрузки товара, работу с заявками от клиентов или закупки материалов, с точки зрения описания системы. Эта нотация помогает понять, из чего должна состоять система, что нужно для автоматизации бизнес-процесса. Но DFD не является описанием непосредственно бизнес-процесса.

Здесь, например, нет такого важного параметра, как время. Также в этой нотации не предусмотрены условия и «развилки». В DFD мы рассматриваем откуда появляются данные, какие данные нужны, их обработку и куда результаты отправить. Т.е. в этой нотации описывается не столько непосредственно процесс, сколько движение потоков данных.

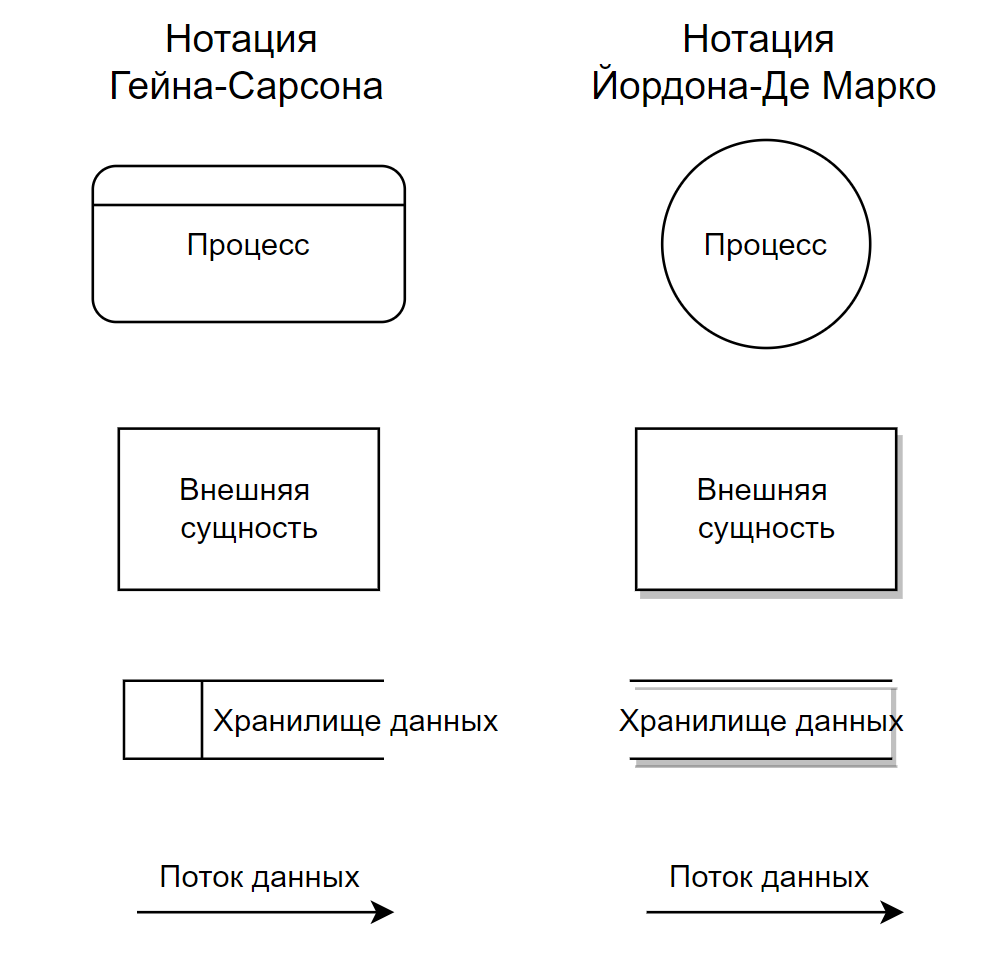


Рисунок 1.4 – Элементы нотации DFD

IDEF0 - используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающие эти функции.

Функциональная модель IDEF0 представляет собой набор блоков, каждый из которых представляет собой процесс со входами и выходами, управлением и механизмами, которые детализируются (декомпозируются) до необходимого уровня. Наиболее важная функция расположена в верхнем левом углу. А соединяются функции между собой при помощи стрелок и описаний функциональных блоков. При этом каждый вид стрелки или активности имеет собственное значение. Данная модель позволяет описать все основные виды процессов, как административные, так и организационные.

Стрелки могут быть (рисунок 1.5):

* Входящие — вводные, которые ставят определенную задачу;
* Исходящие — выводящие результат деятельности;
* Управляющие (сверху вниз) — механизмы управления;
* Механизмы (снизу вверх) — что используется для того, чтобы произвести необходимую работу.

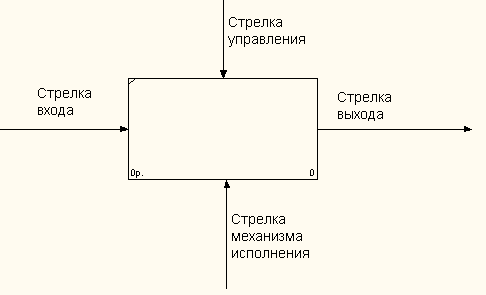


Рисунок 1.5 – Стрелки IDEF0 нотации

Входящие и исходящие стрелки точнее было бы называть вводящими и выводящими, так как по-английски они называются Input и Output соответственно. Но особенности перевода и привычные названия выглядят уже так, как сложилось. И все же для правильного понимания терминов важно помнить их значение в данном случае. Это подтверждается еще и тем, что данная нотация создана прежде всего для разработки ПО, и термины переводить правильнее в этой точки зрения.

Стрелки подписываются при помощи имен существительных (опыт, план, правила), а блоки – при помощи глаголов, т.е. в них описываются действия, которые производятся (создать товар, заключить договор, произвести отгрузку).

Также использовалось бесплатное приложение для моделирования диаграмм бизнес-процессов - Draw.io. С его помощью можно создать практически любое схематическое изображение. Библиотека шаблонов содержит десятки фигур, сгруппированных по категориям. Объекты можно форматировать, изменяя шрифт, цвет, градиент, ширину линии, уровень прозрачности. Доступные форматы экспорта: PDF, GPG, SVG, XML и JPG. Будет использована для проектирования ERD диаграммы.

ERD - Диаграммы "сущность–связь" предназначены для разработки моделей данных и обеспечивают стандартный способ определения данных и отношений между ними. Фактически с помощью ERD осуществляется детализация хранилищ данных проектируемой системы, а также документируются сущности системы и способы их взаимодействия, включая идентификацию объектов, важных для предметной области (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей).

Эти диаграммные техники используются, прежде всего, для проектирования реляционных баз данных (хотя также могут с успехом применяться и для моделирования иерархических и сетевых баз данных).

Диаграммы "сущность-связь" включают:

* Сущность — это класс однотипных объектов, информация о которых должна быть учтена в модели. Каждая сущность должна иметь наименование, выраженное существительным в единственном числе;
* Атрибут — это именованная характеристика, являющаяся некоторым свойством сущности. Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными);
* Связь — это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Каждая связь может иметь один из следующих типов связи (рисунок 1.3):

* Связь типа один-к-одному (1-1) означает, что один экземпляр первой сущности связан с одним экземпляром второй сущности. Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две;
* Связь типа один-ко-многим (1-М) означает, что один экземпляр первой сущности связан с несколькими экземплярами второй сущности. Это наиболее часто используемый тип связи. Левая сущность (со стороны "один") называется родительской, правая (со стороны "много") – дочерней;
* Связь типа много-ко-многим (М-М) означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи много-ко-многим является временным типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа один-ко-многим путем создания промежуточной сущности.

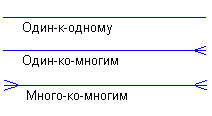


Рисунок 1.4. – Типы связей ERD диаграмм

### 1.3.2 Серверная часть

Практическая реализация информационной системы подразумевает использование языка программирования, а также выбор наиболее подходящей архитектуры информационной системы. В нашем случае, автоматизация работы диспетчера предполагает работу с сервером для хранения информации через визуальный интерфейс. Таким образом, необходимо определить: программный модуль реализации серверной части, библиотека визуального интерфейса и язык программирования.

Рассмотрим самые популярные модули для серверной части:

1. **ASP.NET** - технология для создания веб-приложений на платформе .NET, развиваемая компанией Microsoft. В качестве языков программирования для разработки приложений на ASP.NET используются C# и F#. Основными преимуществами являются: межплатформенная поддержка, модульная система компонентов, встроенное введение зависимостей, управление параллельными версиями и огромное множество дополнительных инструментов. К недостаткам относят довольно трудное масштабирование, а также дороговизну разработки.
2. **Django** - фреймворк для быстрой разработки бэкэнд-сервисов и приложений на Python. Это значит, что с ним можно будет собрать готовый сайт или веб-приложение быстрее, проще и аккуратнее, чем писать весь код самому с нуля. Для этого разработчику сразу доступны: встроенный веб-сервер, механизм авторизации, встроенная система кэширования, шаблоны функций. Из недостатков основным является использование Django ORM, что замедляет разработку и требует изучения дополнительных программных модулей.

Для реализации серверной части был выбран ASP.NET фреймворк, причиной послужил факт наличия модульной системы компонентов, а также наличие дополнительных инструментов.

### 1.3.3 Клиентская часть

Клиентская часть содержит визуальный интерфейс для интуитивного взаимодействия с серверной частью и соответствующую логику.

Рассмотрим самые популярные библиотеки для клиентской части:

1. WinForms - это бесплатная библиотека графических классов с открытым исходным кодом, входящая в состав Microsoft .NET, .NET Framework или Mono, предоставляющая платформу для написания клиентских приложений для настольных компьютеров, ноутбуков и планшетных ПК. Не имеет кроссплатформенности, каждая форма состоит из стандартных элементов Windows, является устаревшим фреймворком. Из плюсов – простота разработки.
2. WPF - Технология WPF (Windows Presentation Foundation) является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов. Основное преимущество WPF - в том, что для отрисовки графики используется не GDI/GDI+, а DirectX. Из явных достоинств WPF можно выделить следующее: 1. WPF поддерживает гибкий поток, размещающий элементы управления на основе их содержимого вместо того что бы фиксировать элементы управления на одном постоянном месте с какими-то постоянными определёнными координатами, поддерживает аудио и видео, платформа не нуждается ни в каких таймерах что бы форма перерисовала себя. Из недостатков – сложность освоения.

Для реализации информационной системы был выбран WPF за счет возможностей перерисовки формы, а также более удобной фиксации визуальных элементов. В связи с выбором ASP.NET и WPF единственным доступным языком программирования, который будет использован для создания информационной системы, является C#.

### 1.3.4 Среда программирования

Из самых популярных средств программирования для языка C# можно выделить:

1. Microsoft Visual Studio — это интегрированная среда разработки (IDE), разработанная и продаваемая Microsoft. Используется для разработки операционных систем Microsoft Windows, драйверов устройств Windows, служб Windows, прикладного программного обеспечения (настольных приложений, приложений UWP, мобильных приложений), веб-сайтов, веб-приложений, веб-служб и многого другого. Дополнительно устанавливая различные компоненты, подготовленные в качестве опций, можно кросс-разрабатывать приложения для других платформ, таких как Linux и Android, в дополнение к платформе Windows.
2. Visual Studio Code — это инструмент для верстальщиков и программистов, который является одним из самых популярных в своей нише. Это редактор кода от компании Microsoft, который является «облегченной» версией их другого популярного продукта — Visual Studio.

Visual Studio Code позиционирует себя как редактор кода, но при помощи плагинов его функциональность можно «разогнать» до уровня полноценной среды разработки. Facebook недавно объявил, что его разработчики используют этот редактор в качестве основного. Доверие такой компании этому программному обеспечению говорит о многом.

1. Project Rider — это быстрая и мощная кросс-платформенная IDE для .NET позволяет открывать, редактировать, собирать, запускать и отлаживать большинство .NET-приложений: дестопные и веб-приложения, библиотеки и сервисы (пока что не поддерживается отладка приложений UWP). Также поддерживаются приложения, разрабатываемые на .NET Core и Mono, в том числе игры на движке Unity и мобильные приложения Xamarin.

Rider поддерживает множество языков .NET-разработки, включая C#, F#, VB.NET, ASP.NET (механизмы визуализации ASPX и Razor), XAML, XML, JavaScript, TypeScript, JSON, HTML, CSS, SCSS, LESS и SQL.

Из всех представленных вариантов, Visual Studio выглядит наиболее подходящими под текущие задачи за счет наличия большого количества дополнительных плагинов и кроссплатформенности.

### 1.3.5 Иные вспомогательные программные средства

Система контроля версий — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Чаще всего в системах контроля версий хранятся исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Система контроля версий даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь системой контроля версий, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими. В данной работе используется Git — распределённая система контроля версий.

Базовые понятие при работе с GIT:

* Репозиторий — место, где система контроля версиями хранит свои метаданные и базу данных объектов проекта, можно сказать, что это основная директория проекта;
* Рабочий каталог — извлечённая из репозитория копия определённой версии проекта;
* Область подготовленных файлов — служебный файл, содержащий информацию о том, что должно войти в следующую ревизию проекта;
* Ревизия — объект, хранящий определенную версию проекта;
* Коммит — ревизия проекта с определенным набором изменений.

В целом алгоритм работы с Git можно описать следующим образом:

Создается Git репозиторий, инициализируется проекта.

Каждый раз, когда проект достигает состояния, которое необходимо сохранить (обычно рекомендуют фиксировать каждое атомарное изменение, т.е. функцию, класс или законченный алгорим), фиксируются “снимки” состояния этих изменений в репозитории.

Каждый файл в вашем рабочем каталоге может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые). Отслеживаемые файлы — это те файлы, которые были в последнем снимке состояния проекта; они могут быть неизменёнными, изменёнными или подготовленными к коммиту. Неотслеживаемые файлы — это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний слепок состояния и не подготовлены к коммиту. При первом клонировании или инициализации репозитория все файлы будут отслеживаемыми и неизменёнными.

При любом редактировании файлов Git будет рассматривать их как изменённые, т.к. их хэш-сумма изменилась со времени последнего коммита. Изменения индексируются, после чего коммит сохраняется и, если необходимо, выгружается на удаленный сервер.

При наличии удаленного сервера и копировании репозитория, команда clone автоматически добавляет этот удалённый репозиторий под именем origin.

Почти каждая система контроля версиями имеет в какой-то форме поддержку ветвления. Ветвление означает, что вы отклоняетесь от основной линии разработки и продолжаете работу, не вмешиваясь в основную линию. Во многих системах это в некотором роде дорогостоящий процесс, зачастую требующий от вас создания новой копии каталога с исходным кодом, что может занять продолжительное время для больших проектов.

В отличии от большинства систем контроля версиями, способ ветвления в Git чрезвычайно легковесен, что делает операции ветвления практически мгновенными и переключение туда-сюда между ветками обычно так же быстрым. Git поощряет процесс работы, при котором ветвление и слияние осуществляется часто, даже по несколько раз в день. Понимание и владение этой функциональностью даёт вам уникальный мощный инструмент и может буквально изменить то, как вы ведёте разработку.