**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО»**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Объектно-ориентированное программирование»

на тему: *Информационная система распространения билетов для автовокзала*

Исполнитель: студент гр. ИТП-21

Гаврильченко А.Е.

Руководитель: преподаватель

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2022

**СОДЕРЖАНИЕ**

[Введение 4](#_Toc103637408)

[1 Обзор литературы по программно-техническому обеспечению 5](#_Toc103637409)

[1.1 Язык программирования *C*# 5](#_Toc103637410)

[1.2 Языкразметки *XML* 8](#_Toc103637411)

[1.3 Язык запросов *LINQ* 9](#_Toc103637412)

[1.4 Принципы *SOLID* 11](#_Toc103637413)

[1.5 Графический пользовательский интерфейс 12](#_Toc103637414)

[1.6 Паттерны проектирования 13](#_Toc103637415)

# **ВВЕДЕНИЕ**

В наш век люди все чаще обращаются к онлайн платформам и мобильным приложениям для приобретения билетов, не исключение составляют и билеты на путешествия автобусом. Электронная система заказов заметно оптимизирует и ускоряет процесс покупки.

Создание приложения, которое структурирует данные и учитывает доступные/проданные билеты на автобус, становится актуальной и важной задачей. Такое решение дает значительное облегчение труда сотрудников, занимающихся продажами и обслуживанием пассажиров, позволяя эффективно управлять информацией о билетах.

Разработка подобного приложения уже давно не является новинкой в сфере обслуживания, поскольку востребованность в улучшении учета данных возникла по причине объема информации, который нужно надежно хранить, постоянно обновлять и организовывать. Разработанное приложение предлагает возможность пользователям выбирать конкретное место в автобусе при покупке билета, что создает наглядное представление об устройстве мест в салоне.

Для хранения информации используются *XML*-файлы, с доступом к данным через язык запросов *Language Integrated Query (LINQ*), что обеспечивает простой и эффективный доступ к данным при работе с приложением.

Для разработки приложения принят объектно-ориентированный язык программирования *C#*, и применение принципов *SOLID* обеспечивает гибкость, расширяемость и адаптивность приложения.

Для построения графического интерфейса использовалась платформа *Windows Form*, предоставляющая удобную среду для создания клиентских приложений.

Проверку функциональности и правильности выполнения задач приложения производили с помощью юнит-тестирования. Это необходимо для обеспечения надежности программы и соответствия ее функционала установленным требованиям.

# **1** **ОБЗОР ЛИТЕРАТУРЫ ПО ПРОГРАММНО-ТЕХНИЧЕСКОМУ ОБЕСПЕЧЕНИЮ**

## **Язык программирования *C#***

*C#*, как язык программирования, занимает лидирующие позиции среди мощных, динамично развивающихся и востребованных технологий в сфере информационных технологий. Этот универсальный язык применяется для создания самых разнообразных приложений: от компактных десктопных программ до масштабных веб-порталов и сервисов, обслуживающих огромное количество пользователей ежедневно.

*C#* имеет синтаксис, схожий с Си, и наследован от *С++* и *Java*. Этот объектно-ориентированный язык программирования унаследовал многое от *Java* и *С++*, включая полиморфизм, наследование, перегрузку операторов и статическую типизацию. Объектно-ориентированный подход *C*# позволяет разрабатывать крупные, гибкие, масштабируемые и легко расширяемые приложения. Постоянное обновление *C*# привносит новые интересные функции с каждой новой версией, такие как лямбды, динамическое связывание, асинхронные методы и другие.

Программы на *C*# выполняются в .*NET*, виртуальной среде выполнения, которая вызывает общеязыковую среду выполнения (*CLR*) и набор классов библиотек .*NET*. *CLR* представляет собой реализацию общеязыковой инфраструктуры (*CLI*), международного стандарта, определенного *Microsoft*. *CLI* является основой для создания сред выполнения и разработки, где языки и библиотеки могут взаимодействовать между собой

На *C*# исходный код компилируется в промежуточный язык (*IL*), соответствующий *CLI*. Код *IL* и ресурсы, такие как изображения и текстовые данные, хранятся в сборке с расширением .*dll*. Сборка содержит метаданные о типах, версиях, языках и других параметрах для сборки.

Во время выполнения программы на *C*# сборка загружается в среду *CLR*, которая выполняет *JIT*-компиляцию из *IL* в машинный код. *CLR* также управляет автоматической сборкой мусора, обработкой исключений и управлением ресурсами. Код, выполняемый *CLR*, иногда называют «управляемым кодом», чтобы подчеркнуть различия с «неуправляемым кодом», который компилируется непосредственно в машинный код для конкретной платформы.

В начале знакомства с языком программирования C# отлично подойдет официальная документация на сайте *Microsoft*, структурированная по различным категориям. Здесь вы найдете разделы как "Начало работы", "Учебники", "Основные понятия" и многое другое. Эта документация представляет собой интерактивное пособие с примерами, которые читатель может выполнять параллельно с чтением. Каждая статья включает ссылки на дополнительные ресурсы, для более глубокого изучения темы. Также для удобства пользователей предлагается возможность скачать электронную версию документации. Для тех, кто заинтересован преимущественно в практических задачах, на сайте *Microsoft* есть бесплатные курсы по *C*#.

Полезным также является сайт *Metanit* по программированию. Здесь вы найдете огромное количество информации не только о *C*#, но и о других популярных языках программирования, а также о различных системах управления базами данных. Раздел, посвященный *C*#, содержит разнообразную информацию. Например, в разделе "Общая тематика" доступно полное руководство по *C*# 9 и .*NET* 5, различные шаблоны проектирования, сетевое программирование в *C*# и .*NET*, алгоритмы, структуры данных. По завершении курса просмотра материала предлагается пройти тесты для проверки усвоенных знаний. Разработка на *C*# охватывает разные области: веб-разработку, мобильные приложения, игры. Metanit также предоставляет информацию о базах данных.

Теперь давайте рассмотрим особенности языка *C*# и их применение в ходе выполнения курсовой работы. Паттерн в программировании представляет собой определенный метод построения кода для решения типовых проблем проектирования. Он предполагает наличие шаблонных проблем, для которых паттерны предлагают ряд принципов решения этих задач.

Разнообразие паттернов программирования предлагает различные решения задач и выполняет разнообразные функции. Анализ конкретных категорий паттернов базируется на назначении или поставленной цели паттерна.

Одной из ключевых категорий паттернов являются порождающие паттерны, которые управляют процессом создания классов и объектов. В данной категории выделяются следующие паттерны:

– Абстрактная фабрика (*Abstract Factory*): обеспечивает интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами, без привязки к конкретным типам данных объектов;

– Строитель (*Builder*): инкапсулирует создание объекта, позволяя разделить его создание на различные этапы;

– Фабричный метод (*Factory Method*): предоставляет интерфейс для создания объектов класса, оставляя решение о создании конкретных объектов на подклассах;

– Прототип (*Prototype*): дает возможность создавать объекты на основе ранее созданных прототипов, предлагая метод клонирования объектов;

– Одиночка (*Singleton*): гарантирует, что для определенного класса будет создан только один объект, а также предоставляет к нему доступ.

Из вышеописанных паттернов, наиболее широко используемым и популярным является паттерн «Одиночка». Однако помимо порождающих паттернов существуют структурные и поведенческие паттерны, которые также играют важную роль в разработке программного обеспечения.

Акцент на принципе *KISS* (*Keep It Simple, Stupid*) при написании кода следует трактовать как сохранение простоты и понятности программного кода. Соблюдение этого принципа позволяет делать код ясным и легким для понимания.

Важнейший атрибут объектно-ориентированного программирования – это наследование. Этот механизм позволяет создавать дочерние классы, которые наследуют, расширяют или изменяют возможности родительского класса. Базовый класс, чьи члены наследуются, называется родительским, а класс, который наследует члены базового класса, называется производным. Наследование часто применяется в проектах, упрощая структуру кода и избавляя от повторения одних и тех же полей и методов в каждом классе. Например, класс Product может служить основой для производных классов Chair, Table, Cupboard и других.

Также *C#* предоставляет ссылочные типы, которые могут иметь значение *NULL*, а также ссылочные типы, не допускающие значение *NULL*, утверждая определенные свойства переменных ссылочного типа. В данном проекте использование ссылочных типов, позволяющих значение *NULL*, определяет принадлежность продукта к группе. Например, определение типа гарнитуры, к которой принадлежит товар, или возврат значения *NULL* при отсутствии принадлежности.

Обработка исключений в *C#* помогает управлять непредвиденными ситуациями, которые возникают во время выполнения программы. Для этой цели используются ключевые слова *try*, *catch* и *finally*, которые позволяют обрабатывать действия, которые могут завершиться неудачно. Например, при ошибке ввода данных, проблемах с доступом к базе данных или некорректном создании объекта вызывается исключение, указывающее на возникшую проблему.

*LINQ (Language Integrated Query)* представляет собой набор технологий, интегрирующих запросы прямо в код на *C#.* Технологии *LINQ* упрощают создание запросов данных, предоставляя удобные выражения, совместимые с классами, методами и событиями. В приложении *LINQ* используется для обработки данных, обеспечивая удобный синтаксис для фильтрации, сортировки и других манипуляций с элементами данных.

*Microsoft* разработал язык запросов *LINQ*, предоставляющий удобный синтаксис для запросов данных. Этот язык позволяет легко фильтровать, сортировать и преобразовывать данные, делая всю операцию более элегантной и понятной. В использованном приложении *LINQ* активно применяется для работы с данными.

## **1.2 Язык разметки *XML***

XML, или расширяемый язык разметки, представляет собой спецификацию, описывающую XML-документы и частично описывающую поведение XML-процессоров. Этот язык разрабатывался для простого форматирования документов, удобного как для человека, так и для программ. Его отличительной чертой является расширяемость, позволяющая разработчику создавать разметку по своим потребностям без жесткой фиксации формата.

С точки зрения физической структуры, документ состоит из сущностей, каждая из которых может ссылаться на другую. В документе существует только один корневой элемент — документная сущность, в то время как его содержание представлено символами.

С логической точки зрения, документ содержит комментарии, объявления, элементы, ссылки на сущности и инструкции обработки. Эти компоненты структурируются разметкой, определяя структуру документа.

Элементы документа должны быть правильно вложены друг в друга. Любой элемент, начинающийся внутри другого, должен завершаться там же. Символьные данные могут содержаться как непосредственно в элементах, так и в специальных секциях "CDATA". Для связывания текста с логическими единицами используются атрибуты, представляющие пары имя-значение.

Имена в XML должны начинаться с буквы, подчеркивания или двоеточия, содержать только допустимые символы, образующие слова Unicode, арабские цифры и некоторые символы. Имена не могут начинаться с "xml" (любым регистром), зарезервированным для W3C. За пределами ASCII можно использовать родной язык для именования.

**Преимущества использования XML**

1) Расширяемость: XML позволяет разработчику создавать собственную разметку в соответствии с требованиями конкретной области.

2)Удобство для человека и программ: Благодаря простому формальному синтаксису, XML удобен как для человека, так и для программного обеспечения.

3)Базирование на Юникод: Использование кодировок Юникод обеспечивает представление содержимого документов, что способствует широкому использованию XML в различных приложениях.

4)Обработка комментариев и инструкций: XML позволяет включать комментарии и инструкции в документы для улучшения их понимания и обработки.

5)Пролог и корневой элемент: Документ начинается с пролога, содержащего объявления и инструкции, а обязательным элементом является корневой элемент, определяющий основную структуру документа.

6)Символьные данные и CDATA: XML обеспечивает способы включения символьных данных в документ, как прямо в элементы, так и с помощью расширенных CDATA-секций.

## **1.3 Язык запросов *LINQ***

LINQ (Language Integrated Query) представляет собой инновационный проект от Microsoft, направленный на улучшение синтаксиса языка запросов в рамках платформы .NET Framework. Этот мощный инструмент, представляющий собой некий симбиоз функциональности SQL и возможностей языков программирования, переносит удобство и выразительность запросов на новый уровень.

LINQ не появился на свет просто так. Его идеи и концепции тесно связаны с предшествующим исследовательским проектом Microsoft Cω (C Omega), где многие из важнейших идей, заложенных в LINQ, были экспериментально испытаны и приведены к жизни. Разработчики Cω сумели подготовить почву для LINQ, что позволило новому проекту сразу выйти на более зрелую стадию и начать широкомасштабное применение.

LINQ вышел в свет вместе с Visual Studio 2008, внедряя в среду разработки потрясающие возможности запросов. Этот интегрированный инструмент позволяет создавать и отлаживать запросы, напоминающие SQL, что существенно упрощает рутинную работу с данными в различных источниках. Не случайно для быстрого прототипирования и тестирования запросов LINQ родилась удивительная утилита под названием LINQPad, ставшая незаменимым помощником многих разработчиков.

LINQ изначально поддерживал запросы к коллекциям объектов в памяти, реляционным базам данных и данным в формате XML. Однако, его дух расширяемости и адаптивности делает этот инструмент универсальным. Разработчики могут расширить возможности LINQ, создав специализированные интерфейсы и провайдеры для работы с собственными источниками данных. Это достигается либо через реализацию стандартных операторов запросов с использованием методов расширения, либо через интерфейс IQueryable, оптимизированный для динамического анализа запросов и перевода их в язык источника данных в рантайме.  
LINQ трансформирует запросы  
LINQ демонстрирует силу компилятора, предоставляя эффективные методы построения деревьев запросов и их трансформации в основе контекста программы. Например, LINQ для SQL (DLinq) использует подход, основанный на построении деревьев выражений при помощи компилятора, что позволяет провайдерам баз данных анализировать запросы и трансформировать их в SQL запросы в зависимости от контекста программы.

Нестандартный подход некоторых энтузиастов привел к созданию библиотек для запросов данных к разнообразным источникам, от WMI и RSS до LDAP и коллекций данных [ADO.NET](https://ado.net/). Их жажда экспериментов и стремление расширить горизонты LINQ позволили исследовать возможности этого инструмента в самых разнообразных областях, продолжая эволюцию и инновации в мире запросов данных.

Вовлеченность Microsoft: взгляд в будущее  
В ближайших перспективных версиях LINQ, Microsoft увлеченно работает над LINQ для XML (ранее XLinq) и [ADO.NET](https://ado.net/) vNext, известным как LINQ to Entities. Эти технологии обещают дальнейшие улучшения в области работы с XML и базами данных, углубляя возможности LINQ и расширяя его влияние на различные сферы программирования и разработки.

Завершение и высокий уровень LINQ  
LINQ представляет собой инновационный шаг в области языков запросов, обогащая платформу .NET Framework мощным и удобным средством работы с данными. Его гибкость, расширяемость и перспективы являются прочным фундаментом для создания удивительных приложений и решений, основанных на эффективном анализе и манипуляции информацией при помощи продвинутых запросов.

## **1.4** **Принципы *SOLID***

*SOLID* — это акроним, объединяющий пять фундаментальных принципов, разработанных с целью облегчения создания поддерживаемого, расширяемого и гибкого кода. Давайте погрузимся глубже в каждый из этих принципов, проведя анализ их значение и воздействие на процесс разработки ПО.

**Принцип единственной обязанности (*SRP*)**

Принцип *SRP* определяет, что каждый класс должен иметь только одну обязанность и эта обязанность должна быть полностью инкапсулирована в самом классе. Это поддерживает простоту кода, упрощает его понимание и поддержку, а также позволяет избежать излишней сложности внутри класса.

**Принцип открытости / закрытости (*OCP*)**

Принцип *OCP* стремится сделать программный код открытым для расширения новым функционалом, но при этом закрытым для изменения существующего кода. Это обеспечивает стабильность и облегчает добавление новых возможностей без необходимости внесения изменений в уже работающий функционал.

**Принцип подстановки Барбары Лисков (*LSP*)**

Принцип *LSP* утверждает, что объекты должны быть заменяемы другими экземплярами своих подтипов без изменения корректности программы. Это способствует созданию гибкой системы, где объекты могут взаимозаменяться, не нарушая ее функциональность.

**Принцип разделения интерфейса (*ISP*)**

ISP призывает к разделению интерфейсов на более мелкие и специфические части, чтобы клиенты зависели только от необходимых им методов. Это повышает гибкость и понятность системы, уменьшая зависимость между компонентами.

**Принцип инверсии зависимостей (*DIP*)**

Принцип *DIP* сфокусирован на том, чтобы модули верхнего уровня не зависели от модулей нижнего уровня, оба типа модулей зависели от абстракций. Это способствует уменьшению связанности между различными частями системы, обеспечивая большую гибкость и поддерживаемость.

Применение принципов *SOLID* в проектировании ПО способствует созданию сбалансированных, гибких и масштабируемых систем с простым понятным кодом. Ключевым влиянием этих принципов является снижение сложности, повышение удобства сопровождения и расширения системы, а также обеспечение высокой степени надежности и простоты тестирования. *SOLID* — это не просто набор правил, это философия проектирования, значительно улучшающая качество разрабатываемого программного обеспечения.

## **1.5 Графический пользовательский интерфейс**

(GUI) переводит взаимодействие пользователя с компьютером на уровень интуитивного взаимодействия с помощью визуальных элементов и графических индикаторов. Основное преимущество GUI заключается в его доступности для широкого круга пользователей, независимо от уровня их технических знаний – от новичков до опытных разработчиков и специалистов в области технологий.

GUI позволяет пользователям проводить различные действия, такие как открытие меню, перемещение файлов, запуск приложений и поиск в Интернете, без необходимости запоминания командной строки для взаимодействия с компьютером.

Одним из ключевых аспектов GUI является мгновенная обратная связь, где пользователь немедленно видит результат своих действий. Например, клик по значку открывает его в реальном времени, что создает непосредственную связь между пользователем и системой. В отличие от интерфейса командной строки, пользователь не узнает точно, что произойдет, пока не введет команду и не нажмет Enter.

При разработке графического интерфейса в приложениях широко использовался Windows Forms. Windows Forms является частью Microsoft .NET Framework и предоставляет API для создания пользовательского интерфейса в среде Windows. Этот интерфейс облегчает доступ к элементам интерфейса Windows путем создания обертки для Win32 API в управляемом коде.

Windows Forms рассматривается как замена старой библиотеке MFC и обеспечивает простой доступ к элементам управления Windows за счет инкапсуляции функционала Win32 API в управляемом коде. Windows Forms не только предоставляет удобные средства для создания интерфейса, но и не зависит от языка программирования, что позволяет разработчикам использовать его на различных платформах.

Windows Forms обладает множеством возможностей, таких как создание диалоговых окон, поддержка печати, добавление справочной информации и локализация приложений на разных языках. Более того, Windows Forms обеспечивает надежность и безопасность благодаря интеграции с системой безопасности .NET Framework, что способствует созданию надежных и устойчивых приложений.

## **1.6 Паттерны проектирования**

Паттерны проектирования представляют собой мощные инструменты для решения сложностей, возникающих при проектировании архитектуры программного продукта. Они предлагают структурированные и проверенные подходы к решению определенных проблем.

В отличие от готовых функций или библиотек, паттерны проектирования не представляют конкретный код, а обобщенные рекомендации и абстрактные описания. Они помогают разработчикам создавать гибкие, расширяемые и поддерживаемые программы, где каждый паттерн может быть адаптирован и применен в соответствии с конкретными требованиями проекта.

Паттерны проектирования делятся на различные категории в зависимости от их функциональности и основных применяемых сценариев:

1. Порождающие паттерны предоставляют механизмы для создания и инициализации объектов, предлагая гибкость в выборе и настройке создаваемых объектов.

2. Структурные паттерны помогают организовать отношения и взаимодействия между объектами, облегчая построение сложных структур.

3. Поведенческие паттерны определяют эффективные способы взаимодействия объектов и классов, обеспечивая контроль над их поведением и обязанностями.

Помимо паттернов проектирования, также имеют большое значение репозитории и шаблоны единиц работы. Репозиторий - это промежуточное звено, которое абстрагирует доступ к данным и предоставляет удобный интерфейс для работы с ними. Шаблоны единиц работы, такие как "Unit of Work", способствуют упрощению работы с различными репозиториями и обеспечивают использование единого контекста базы данных.

Реализация этих шаблонов помогает создать правильную структуру для развертывания приложения, обеспечить инверсию зависимостей и применять практики DI (Dependency Injection), что значительно облегчает тестирование и разработку приложений.