Министерство образования и науки

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Отчет

по лабораторной работе № 7

# «Моделирование интерфейса пользователя»

**Программное обеспечение Мини-АТС**

по дисциплине «Архитектура программного обеспечения инфокоммуникационных систем»

**Выполнил: Кислюк И. В. K4220**

**Проверил: к.т.н., доцент Н.А. Осипов**

Санкт-Петербург

2018

## Цель работы.

– освоение процесса моделирования интерфейса пользователя для проектирования архитектуры прикладных программных систем,

– определение требований к программной системе" (Requirements) в соответствии с Rational Unified Process по моделированию интерфейса пользователя.

## Задачи, решаемые при выполнении работы.

### Создание архитектуры приложения.

### Определить элементы, отображаемые на UML-схемах.

## Объект исследования.

Архитектура приложения.

## Метод экспериментального исследования.

Имитационное визуально ориентированное моделирование.

## Проектирование компонентов пользовательского интерфейса и компонентов логики представления

### Шаг 1 - Понимание предъявляемых к UI требований

Требования к UI определяются функциональностью, которую должно поддерживать приложение, и ожиданиями пользователей.

Начните с выяснения, кто будет пользователями приложения, и понимания целей и задачей, которые эти пользователи желают реализовывать при использовании приложения.

Особое внимание уделите вопросу последовательности задач или операций; выясните, ожидают ли пользователи структурированного последовательного взаимодействия или неструктурированного взаимодействия с произвольным порядком операций, когда существует возможность выполнения множества задач одновременно. Как часть этого процесса, также выясните, какие данные понадобятся пользователям, и формат, в котором они ожидают их увидеть. Возможно, придется провести исследования, чтобы лучше понять среду, в которой пользователь будет взаимодействовать с приложением. Кроме того, рассмотрите текущие уровни взаимодействия с пользователем и сравните их с требованиями к взаимодействию с пользователем, предъявляемыми для разрабатываемого UI, чтобы убедиться в их логичности и понятности. Все эти факторы помогут создать ориентированный на пользователя дизайн.

Один из факторов, имеющих большое влияние на выбор технологии - требуемая функциональность UI. Выясните, должен ли UI предоставлять насыщенную функциональность или взаимодействие с пользователем, должен ли он обеспечивать минимальное время отклика или требует графической или анимационной поддержки. Также рассмотрите требования с точки зрения локализации к типам данных, форматам и форматам представления для таких данных, как даты, время и валюты. Кроме того, определите требования по персонализации приложения, такие как предоставление пользователю возможности менять компоновку и стили во время выполнения.

Чтобы сделать UI интуитивно понятным и простым в использовании, продумайте компоновку или композицию интерфейса, а также перемещение пользователя по UI приложения. Это поможет выбрать соответствующие элементы управления и технологии для UI. Разберитесь с тем, какие требования физического устройства отображения (такие как размер и разрешение экрана) и специальные возможности (такие как крупный текст или кнопки, рукописный ввод и т.д.) необходимо поддерживать. Примите решение о том, как будете выполнять группировку взаимосвязанных данных в разделах UI, избегать конфликтов или неоднозначностей интерфейса и выделять важные элементы. Обеспечьте пользователям возможность быстро и легко находить сведения в приложении посредством навигационных элементов управления, функций поиска, четко именованных разделов, карт сайта и других соответствующих возможностей.

### Шаг 2 - Выбор необходимого типа UI

На основании предъявляемых к UI требований можно принять решение о типе UI для приложения.

Существует ряд разных типов UI, каждый из которых обладает определенными преимуществами и недостатками. Часто обнаруживается, что предъявляемым к UI требованиям соответствует несколько типов UI. Но бывают ситуации, когда ни один из типов UI не обеспечивает полностью все требования. В этом случае необходимо рассмотреть возможность создания нескольких разных типов UI, которые будут совместно использовать бизнес-логику. Примером этому может служить приложение для call-центра, некоторые из возможностей которого предоставляются клиенту для самостоятельного использования через Веб и на мобильных устройствах.

Мобильные приложения могут разрабатываться как тонкое клиентское или насыщенное клиентское приложение. Насыщенные клиентские мобильные приложения могут поддерживать сценарии без подключения или с периодическим подключением. Веб или тонкие клиентские мобильные приложения поддерживают только сценарии с подключением. Ограничением при проектировании мобильных приложений могут быть и аппаратные ресурсы.

Насыщенные клиентские приложения обычно являются автономными или сетевыми приложениями с графическим пользовательским интерфейсом, отображающим данные с помощью различных элементов управления, развертываемыми на настольном или портативном компьютере локального пользователя. Эти приложения подходят для сценариев без подключения или с периодическим подключением, поскольку выполняются на клиентском компьютере. Насыщенное клиентское приложение является хорошим выбором, если требуется высокодинамичный UI с малым временем отклика или UI должен обеспечивать насыщенные функциональность и взаимодействие с пользователем; либо если приложение должно поддерживать как сценарии с подключением, так и сценарии без подключения, использовать ресурсы локальной системы на клиентском компьютере или интегрироваться с другими приложениями на этом компьютере.

Обычно насыщенные Интернет-приложения (RIA) – это Веб-приложения с насыщенным графическим пользовательским интерфейсом, выполняющиеся в браузере. Как правило, приложения RIA используются в сценариях с подключением. Используйте RIA, если необходимо обеспечить UI, поддерживающий динамическое взаимодействие с пользователем с малым временем отклика или использующий потоковое мультимедиа и доступный на широком диапазоне устройств и платформ. Эти приложения могут использовать вычислительные мощности клиентского компьютера, но не могут напрямую взаимодействовать с локальными ресурсами системы, такими как веб-камеры (начиная с Silverlight 4.0, RIA приложения могут работать с камерой и микрофоном), или с другими клиентскими приложениями, такими как приложения Microsoft Office.

Веб-приложения поддерживают сценарии с постоянным подключением и могут поддерживать множество разных браузеров, выполняющихся под управлением множества различных операционных систем и на разных платформах. Веб-приложение - замечательный выбор, если UI должен быть стандартизованным, доступным для широчайшего диапазона устройств и платформ и работать только при постоянном подключении к сети. Также Веб­приложения хорошо подходят, если необходимо обеспечить доступность содержимого приложения для поиска средствами поиска в Веб.

Консольные приложения предлагают альтернативный текстовый пользовательский интерфейс и обычно выполняются в командных оболочках, таких как Command window (Интерфейс командной строки) или Power Shell. Такие приложения лучше всего подходят для задач администрирования или разработки и не используются как часть многослойного дизайна приложений.

### Шаг 3 - Выбор технологии UI

После того, как вы определились с типом UI для своих компонентов UI, необходимо выбрать соответствующую технологию.

Как правило, выбор зависит от выбранного типа UI. Далее рассмотрим, какие технологии подходят для каждого из типов UI:

Пользовательские интерфейсы мобильных клиентов могут быть реализованы с использованием следующих технологий создания пользовательского интерфейса:

* Microsoft .NET Compact Framework. Это версия Microsoft .NET Framework, разработанная специально для мобильных устройств. Используйте эту технологию для мобильных приложений, которые должны выполняться на устройствах без гарантированного подключения к сети.
* ASP.NET для мобильных устройств. Это версия ASP.NET, разработанная специально для мобильных устройств. ASP.NET для мобильных приложений может размещаться на сервере Internet Information Services (IIS, Информационные Интернет-службы. Используйте эту технологию для мобильных Веб-приложений, если требуется поддерживать широкий диапазон мобильных устройств и браузеров и можно рассчитывать на постоянное подключение к сети.
* Silverlight для мобильных устройств. Для этой версии Silverlight-клиента требуется, чтобы на мобильном устройстве был установлен подключаемый модуль Silverlight. Используйте эту технологию, чтобы портировать существующие Silverlight- приложения на мобильные устройства, или если желаете создавать более насыщенные UI, чем обеспечивают другие технологии.

Пользовательские интерфейсы насыщенных клиентов могут быть реализованы с использованием следующих технологий представления:

* Windows Presentation Foundation (WPF). Приложения WPF поддерживают более широкие графические возможности, такие как 2-D и 3-D графика, независимость от разрешения экрана, расширенная поддержка документов и полиграфического оформления, анимация с временной шкалой, потоковое аудио и видео и векторная графика.
* Windows Forms. Windows Forms является частью .NET Framework с момента ее появления и идеально подходит для бизнес-приложений. Даже несмотря на существование возможностей Windows Presentation Foundation (WPF), Windows Forms будет идеальным решением для приложений, к которым не предъявляются никакие особые требования по насыщенным медиа или интерактивным возможностям, или если группа разработки уже имеет богатый опыт работы с Windows Forms.
* Windows Forms с элементами WPF. Этот подход позволяет использовать преимущества, предоставляемые элементами управления WPF, для создания более мощных UI. WPF можно добавлять в существующее приложение Windows Forms, возможно, как этап постепенного перехода к реализации полностью на WPF. Используйте этот подход для введения насыщенных медиа и интерактивных возможностей в существующие приложения, но не забывайте, что элементы управления WPF лучше всего работают на мощных клиентских компьютерах.
* WPF с элементами Windows Forms. Этот подход позволяет дополнить WPF элементами управления Windows Forms, предоставляющими функциональность, не обеспечиваемую WPF. Добавлять элементы управления Windows Forms в UI можно с помощью элемента WindowsFormsHost из сборки WindowsFormsIntegration. Используйте этот подход, если в пользовательском интерфейсе WPF необходимы элементы управления Windows Forms, но не забывайте о некоторых ограничениях и сложностях, связанных с перекрытием элементов управления, фокусом интерфейса и методиками формирования визуального представления, используемыми разными технологиями.
* XAML Browser Application (XBAP), использующее WPF. Данная технология позволяет размещать WPF-приложение в изолированной программной среде в Microsoft Internet Explorer или Mozilla Firefox для Windows. В отличие от Silverlight, в данном случае, доступны все возможности инфраструктуры WPF лишь с некоторыми ограничениями относительно доступа к системным ресурсам из частично доверяемой изолированной программной среды.

Пользовательские интерфейсы насыщенных Интернет-приложений могут быть реализованы с использованием следующих технологий представления:

* Silverlight. Это оптимизированный для работы в браузере аналог WPF, не зависящий от платформы и браузера. По сравнению с XBAP, Silverlight меньше и быстрее устанавливается. Благодаря своему небольшому размеру и поддержке разных платформ Silverlight является хорошим выбором для графических приложений, не требующих полной поддержки графических возможностей WPF, или в случае, когда необходимо избежать установки приложения на клиенте.
* Silverlight с AJAX. Silverlight поддерживает Асинхронный JavaScript и XML (Asynchronous JavaScript and XML, AJAX) и предоставляет свою объектную модель для доступа из JavaScript, размещаемого в Веб-странице. Эту возможность можно использовать для обеспечения взаимодействия между компонентами Веб-­страницы и приложением Silverlight.

Пользовательские интерфейсы Веб-приложений могут быть реализованы с использованием следующих технологий формирования представления:

* ASP.NET Web Forms. Это фундаментальная технология проектирования и реализации UI для Веб-приложений .NET. Приложение ASP.NET Web Forms должно быть установлено только на Веб-сервере, на клиентский компьютер никакие компоненты устанавливать не надо. Используйте эту технологию для Веб-приложений, в которых не требуются дополнительные возможности, предоставляемые AJAX, Silverlight, MVC или Dynamic Data.
* ASP.NET Web Forms с AJAX. Применяйте AJAX с ASP.NET Web Forms для асинхронной обработки запросов между сервером и клиентом, что сократит время отклика, обеспечит более насыщенное взаимодействие с пользователем и сократит количество обращений к серверу. AJAX входит в состав ASP.NET в .NET Framework версии 3.5 и последующих.
* ASP.NET Web Forms с элементами управления Silverlight. Добавив элементы управления Silverlight в готовое приложение ASP.NET, его можно расширить более насыщенными возможностями создания визуального представления и взаимодействия с пользователем. И это избавит от необходимости создавать совершенно новое Silverlight-приложение. Такой подход хорош для внедрения насыщенного медиа-содержимого Silverlight в существующее Веб-приложение. Элементы управления Silverlight и включающая их Веб-страница могут взаимодействовать на стороне клиента с помощью JavaScript.
* ASP.NET MVC. Эта технология позволяет использовать ASP.NET для создания приложений на базе шаблона Model-View-Controller (MVC). Используйте данную технологию, если требуется поддерживать разработку через тестирование и обеспечить четкое разделение функциональности обработки UI и формирования визуального представления UI. Такой подход также поможет получить полный контроль над формированием HTML и избежать смешения данных представления с кодом логики обработки.
* ASP.NET Dynamic Data. Эта технология позволяет создавать управляемые данными ASP.NET-приложения, использующие модель данных Language-Integrated Query (LINQ) to Entities. Применяйте ее, если необходима модель быстрой разработки управляемых данными LOB-приложений, основанных на простом формировании шаблонов, но при этом поддерживающих полную настройку.

### Шаг 4 - Проектирование компонентов представления

Следующий шаг после выбора технологии реализации UI – проектирование компонентов UI и компонентов логики представления.

Могут использоваться следующие типы компонентов представления:

* компоненты пользовательского интерфейса;
* компоненты логики представления;
* компоненты модели представления.

Эти компоненты поддерживают разделение функциональных областей в рамках слоя представления и часто используются для реализации шаблонов раздельного представления, таких как MVP (Model-View-Presenter) или MVC (Model-View-Controller), через разделение задач обработки UI на три роли: Модель, Представление и Контроллер/Презентатор.

Такое разделение функциональности слоя представления повышает удобство обслуживания, тестируемость и возможности повторного использования. Применение абстрактных шаблонов, таких как внедрение зависимостей, также способствует упрощению тестирования логики представления.

#### Компоненты пользовательского интерфейса

Компоненты UI – это визуальные элементы, отображающие данные пользователю и принимающие пользовательский ввод.

В рамках отдельного слоя представления их обычно называют Представлениями (Views). При проектировании компонентов UI руководствуйтесь следующими рекомендациями:

* Разбейте страницы или окна на отдельные пользовательские элементы управления, чтобы упростить и обеспечить возможность повторного использования этих элементов управления. Выбирайте соответствующие компоненты UI и используйте преимущества возможностей связывания данных элементов управления, применяемых в UI.
* Избегайте создания иерархий наследования пользовательских элементов управления и страниц, чтобы обеспечить возможность повторного использования кода. Отдавайте предпочтение композиции, а не наследованию, и создавайте компоненты логики представления, пригодные для повторного использования.
* Создавайте специализированные элементы управления, только если это необходимо для специализированного отображения или сбора данных.

Если видите, что имеющиеся требования к UI не могут быть реализованы стандартными элементами управления, прежде чем создавать собственные специализированные элементы управления, рассмотрите возможность приобретения готового набора элементов управления.

При создании специализированных элементов управления старайтесь расширять существующие элементы управления, а не создавать элементы управления с нуля. Расширяйте существующие элементы управления путем подключения к ним поведения, а не наследования от них. Реализуйте поддержку дизайнера для специализированных элементов управления, чтобы разработчикам было проще работать с ними.

#### Компоненты логики представления

Компоненты логики представления занимаются не визуальными аспектами пользовательского интерфейса, к которым обычно относится проверка, ответ на действия пользователя, взаимодействие компонентов UI и координирование взаимодействий с пользователем.

Компоненты логики представления необходимы не всегда; создавайте их, только если собираетесь выполнять в слое представления большой объем обработки, которая должна быть отделена от компонентов UI, или если хотите обеспечить более благоприятные условия для модульного тестирования логики представления.

При проектировании компонентов логики представления руководствуйтесь следующими рекомендациями:

* Если UI требует сложной обработки или должен обмениваться данными с другими слоями, используйте компоненты логики представления для отделения этой обработки от компонентов UI.
* Используйте компоненты логики представления для хранения состояния, относящегося к UI, но не характерного для конкретной реализации. Старайтесь не включать в компоненты логики представления бизнес-логику или бизнес-правила, кроме валидации ввода и данных. Также избегайте реализации в компонентах логики представления логики формирования визуального представления UI или специализированной логики UI.
* С помощью компонентов логики представления обеспечьте согласованное состояние пользовательского интерфейса при восстановлении приложения после сбоя или ошибки.
* Там, где UI требует поддержки сложного рабочего процесса, создавайте отдельные компоненты рабочего процесса, использующие такую систему управления рабочим процессом, как Windows Workflow Foundation, или реализуйте собственный механизм в бизнес-слое приложения.

#### Компоненты модели представления

Компоненты модели представления представляют данные, поступающие с бизнес-слоя, в формате, доступном для использования UI и компонентами логики представления. Обычно модели представляют данные, и поэтому используют компоненты доступа к данным и, возможно, компоненты бизнес-слоя для сбора этих данных. Если модель также инкапсулирует бизнес-логику, ее обычно называют сущностью представления. Компоненты модели представления могут, к примеру, агрегировать данные из множества источников, преобразовывать данные для обеспечения удобства их отображения в UI, реализовывать логику проверки и помогать в представлении бизнес-логики и состояния в рамках слоя представления. Обычно они используются для реализации шаблонов раздельного представления, таких как MVP or MVC.

При проектировании компонентов модели представления руководствуйтесь следующими рекомендациями:

* Определитесь, нужны ли вам компоненты модели представления. Обычно модели слоя представления используются для отображения специальных данных или форматов слоя представления или в случае применения шаблона раздельного представления, такого как MVP или MVC.
* Работая с элементами управления с привязкой к данным, проектируйте или выбирайте соответствующие компоненты модели представления, которые можно легко связать с элементами управления UI. При использовании в качестве формата компонента модели представления специальных объектов, коллекций или наборов данных обеспечьте реализацию ими соответствующих интерфейсов и событий для поддержки привязки данных.
* При выполнении валидации данных в слое представления размещайте код валидации в компонентах модели представления. Но также продумайте преимущества использования кода или библиотек для централизованной валидации.
* Рассмотрите требования сериализации для данных, передаваемых в компоненты модели представления, если эти данные будут передаваться по сети или сохраняться на жестком диске клиента.

Также необходимо выбрать подходящий тип данных для компонентов модели представления и сущностей представления. Этот выбор определяется требованиями, предъявляемыми к приложению, и ограничениями, налагаемыми инфраструктурой и возможностями разработки. Начните с выбора формата данных слоя представления и примите решение о том, будут ли компоненты также инкапсулировать бизнес-логику и состояние. Далее необходимо принять решение о том, как будут представляться данные в пользовательском интерфейсе.

Существуют такие общие форматы представления данных:

* Собственный класс. Используйте собственный класс, если необходимо представлять данные как сложный объект, проецируемый непосредственно на бизнес-сущности. Например, можно создать собственный объект Order, который будет представлять данные заказа. Также собственный класс может использоваться для инкапсуляции бизнес-логики и состояния и осуществления проверки в слое представления или реализации собственных свойств.
* Array (Массив) и Collection (Коллекция). Используйте массив или коллекцию, если требуется выполнить привязку данных к элементам управления, таким как окно списка или выпадающий список, в которых используются значения одного столбца.
* DataSet (Набор данных) и DataTable (Таблица данных). Используйте DataSet или DataTable при работе с простыми табличными данными и элементами управления с привязкой к данным, такими как таблица, окно списка и выпадающий список.
* Typed Dataset (Типизированный набор данных). Используйте Typed DataSet, если хотите реализовать тесное связывание с бизнес-сущностями, чтобы избежать возникновения несогласованности из-за изменений базы данных.
* XML. Этот формат полезен при работе с Веб-клиентом, когда данные могут быть встроены в Веб-страницу или извлекаться через Веб-сервис или HTTP-запрос. Выбирайте XML при работе с такими элементами управления, как дерево или таблица. Также XML легко сохранять, сериализовать и передавать по каналам связи.
* DataReader (Модуль чтения данных). Используйте DataReader в сценариях с постоянным подключением для извлечения данных в режиме только для чтения и только для пересылки. DataReader обеспечивает эффективный способ для последовательной обработки данных, поступающих из базы данных, или для извлечения больших объемов данных. Однако он очень тесно связывает логику со схемой базы данных, что, как правило, не рекомендуется.

#### Сущности представления

Компоненты модели представления должны по возможности инкапсулировать и данные, поступающие с бизнес-слоя, и бизнес-логику, и поведение. Это позволяет обеспечить непротиворечивость и корректность данных в слое представления и способствует улучшению качества взаимодействия с пользователем.

В некоторых случая в роли компонентов модели представления могут выступать бизнес-сущности бизнес-слоя, используемые напрямую слоем представления. В других случая компоненты модели представления могут представлять подмножество компонентов бизнес-сущностей, в частности, разработанных для поддержки слоя представления приложения. Например, в них могут храниться данные в формате, более удобном для использования UI и компонентами логики представления. Такие компоненты иногда называют сущностями представления.

Если бизнес-слой и слой представления располагаются на клиенте, что является типовым сценарием для насыщенных клиентских приложений, бизнес-сущности обычно используются напрямую из бизнес-слоя. Однако если необходимо сохранять или обрабатывать бизнес- данные в формате или способом, отличным от формата и поведения, предоставляемыми бизнес-сущностями бизнес-слоя, можно рассмотреть возможность применения сущностей представления.

Если бизнес-слой и слой представления располагаются на разных уровнях, использование бизнес-сущностей уровнем представления может быть реализовано через их сериализацию и передачу по сети с помощью объектов передачи данных с последующим их восстановлением в виде экземпляров бизнес-сущностей на уровне представления. Или можно восстанавливать данные как сущности представления, если требуемый формат и поведения отличаются от используемых бизнес-сущностями. Этот сценарий продемонстрирован на рис. 1.

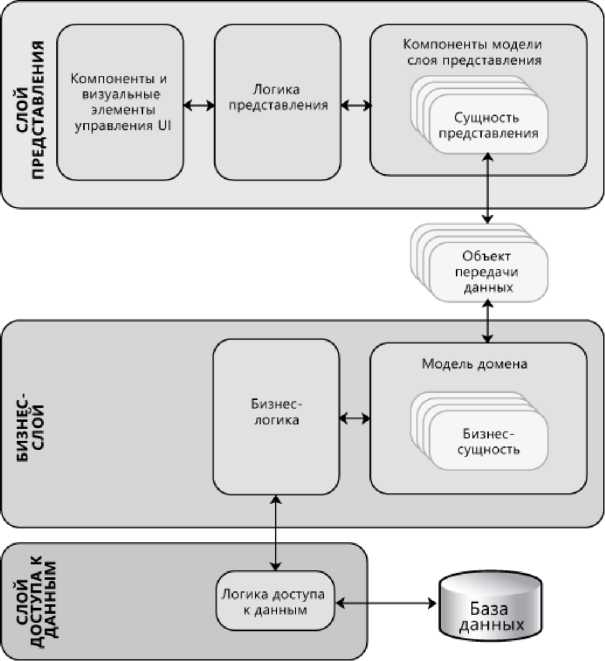


Рис. 1 Компоненты модели представления и сущности представления

Компоненты модели представления и сущности представления могут быть полезны при размещении слоя представления и бизнес-слоя на разных уровнях

### Шаг 5 - Определение требований к привязке данных

Привязка данных обеспечивает возможность создания связи между элементами управления пользовательского интерфейса и данными или логическими компонентами приложения.

Привязка данных позволяет отображать и взаимодействовать с данными баз данных, а также данными других структур, таких как массивы и коллекции.

Привязка данных - это мост между целью привязки (обычно это элемент управления пользовательского интерфейса) и источником привязки (обычно это структура данных, модель или компонент логики представления).

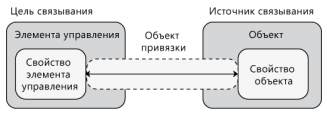


Рис.2. Объекты, используемые при привязке данных

Как показано на рис. 2.4, обычно в привязке данных участвуют четыре элемента, взаимодействующих друг с другом для обновления свойств связанного объекта значениями, предоставляемыми источником привязки. Элементы управления с привязкой к данным - это элементы управления, связанные с источниками данных. Например, элемент управления DataGrid связан с коллекцией объектов. Привязка данных часто используется с шаблонами раздельного представления для связывания компонентов UI (Представлений) с презентаторами или контроллерами (компоненты логики представления) либо с моделью слоя представления или компонентами сущностей.

Поддержка привязки данных и ее реализация в каждой технологии UI разная. В общем, большинство технологий UI позволяют выполнять привязку элементов управления к объектам и спискам объектов. Но некоторые специальные технологий привязки данных могут потребовать реализации в источниках данных определенных интерфейсов и событий для обеспечения полной поддержки привязки данных, например, интерфейса INotifyPropertyChanged (Уведомление об изменении свойства) в WPF или IBindingList (Список привязки) в Windows Forms. При использовании шаблона раздельного представления логика представления и компоненты данных должны гарантированно поддерживать необходимые интерфейсы или события, чтобы обеспечить простоту привязку элементов управления UI к ним.

Обычно используются два типа привязки:

* Односторонняя привязка. Изменения свойства источника приводят к автоматическому обновлению целевого свойства, но изменения целевого свойства не распространяются на исходное свойство. Такой тип привязки подходит для неявно доступных только для чтения элементов управления. Примером такой односторонней привязки могут быть биржевые сводки. Если нет необходимости отслеживать изменения целевого свойства, использование односторонней привязки позволит избежать ненужных издержек.
* Двухсторонняя привязка. Изменения любого из свойств, исходного либо целевого, приводят к автоматическому обновлению второго свойства. Такой тип привязки подходит для редактируемых форм или других полностью интерактивных сценариев UI. Многие редактируемые элементы управления в Windows Forms, ASP.NET и WPF поддерживают двухстороннюю привязку, так что изменения источника данных отражаются в элементе управления UI, и изменения в элементе управления UI отражаются в источнике данных.

### Шаг 6 - Выработка стратегии обработки ошибок

Компоненты UI являются внешней границей приложения, и поэтому должны реализовывать соответствующую стратегию обработки ошибок для обеспечения стабильности приложения и положительного впечатления при взаимодействии с пользователем. При проектировании стратегии обработки ошибок рассмотрите следующие варианты:

* Стратегия централизованной обработки исключений. Обработка исключений и ошибок относится к сквозной функциональности. Она должна реализовываться в отдельных компонентах, которые обеспечивают централизацию этой функциональности и делают ее доступной во всех слоях приложения. Это также упрощает обслуживание и способствует повторному использованию.
* Протоколирование исключений. Очень важно протоколировать ошибки на границах системы, чтобы служба поддержки могла выявлять и диагностировать их. Это важно для компонентов представления, но может создавать большие сложности для кода, выполняющегося на клиентских компьютерах. Будьте осторожны и тщательно выбирайте методы протоколирования информации личного порядка (Personally Identifiable Information, PII) или конфиденциальных данных и обратите особое внимание на размер и размещение журнала.
* Вывод на экран понятных пользователю сообщений. При использовании этой стратегии в случае возникновения ошибки на экран выводится понятное пользователю сообщение с указанием причины ошибки и описанием, как ее можно исправить. Например, ошибки проверки данных должны отображаться так, чтобы было понятно, какие данные являются ошибочными и почему. В этом сообщении также указывается, как пользователь может исправить или ввести действительные данные.
* Разрешение повторной попытки. При использовании этой стратегии на экран выводится понятное пользователю сообщение, объясняющее причину ошибки и предлагающее пользователю повторить операцию. Такая стратегия полезна, если ошибки формируются из-за возникновения временных исключительных ситуаций, таких как недоступность ресурса или истечение времени ожидания сети.
* Вывод на экран универсальных сообщений. Если в приложении возникает непредвиденная ошибка, данные ошибки необходимо запротоколировать, но для пользователя вывести только универсальное сообщение. Предоставьте пользователю уникальный код ошибки, который может быть представлен группе технической поддержке. Эта стратегия полезна при возникновении непредвиденных исключений. Как правило, в случае возникновения непредвиденного исключения рекомендуется закрыть приложение, чтобы предотвратить повреждение данных или риски безопасности.

### Шаг 7 - Определение стратегии валидации

Эффективная стратегия валидации пользовательского ввода поможет фильтровать нежелательные или злонамеренные данные и будет способствовать повышению защищенность приложения.

Как правило, валидация ввода осуществляется слоем представления, тогда как проверка на соответствие бизнес-правилам проводится компонентами бизнес-слоя.

При проектировании стратегии проверки, прежде всего, необходимо определить все вводимые данные, подлежащие проверке. Например, ввод от Веб-клиента в поля формы, параметры (такие как данные операций GET и POST и строки запросов), скрытые поля и состояние представления (view state) подлежат проверке. В общем, проверяться должны все данные, поступающие из источников, не имеющих доверия.

Для приложений, имеющих компоненты и на стороне клиента, и на стороне сервера, таких как приложения RIA или насыщенные клиентские приложения, вызывающие сервисы на сервере приложений, кроме всех проверок на клиенте, должна проводиться дополнительная проверка на сервере. Но для повышения удобства использования и по соображениям производительности некоторые из проверок можно продублировать на клиенте. Проверка на клиенте позволяет обеспечивать пользователям быструю обратную связь в случаях ввода ими некорректных данных. Это может сохранить время и полосу пропускания, но не забывайте, что злоумышленники могут обойти любую реализованную на клиенте проверку.

Определившись с данными, подлежащими проверке, выберите методики проверки для них. Самыми распространенными методиками проверки являются:

* Прием заведомо допустимого (Список разрешенного ввода или позитивная проверка). Принимаются только данные, удовлетворяющие заданным критериям, все остальные данные отклоняются.
* Отклонение заведомо недопустимого (Список запрещенного ввода или негативная проверка). Принимаются данные, не содержащие известный набор символов или значений.
* Очистка. Известные плохие символы или значения устраняются или преобразовываются с целью сделать ввод безопасным.

Рекомендуется принимать заведомо допустимые значения (Список разрешенного ввода), а не пытаться выявить все возможные недействительные или злонамеренные значения, которые должны быть отклонены. Если невозможно полностью определить список известных допустимых значений, можно дополнить проверку частичным списком известных недопустимых значений и/или проводить очистку в качестве второй линии защиты.

Разные технологии представления используют разные подходы к проверке и информированию пользователя о проблемах. В WPF, к примеру, используются конвертеры и объекты правил проверки, часто подключаемые с помощью XAML, тогда как Windows Forms обеспечивает события проверки и привязки.

## Выводы и анализ результатов работы.

Прежде всего, требуется понять, какие требования предъявляются к UI, и суметь выбрать соответствующую технологию. После этого уже можно принимать решения о связывании логики представления и данных с элементами управления UI. Также необходимо иметь четкое представление о требованиях к обработке ошибок и проверке в UI.

Архитектор интерфейса пользователя должен учитывать общий эффект от принимаемых проектных решений, обязательно присутствующие компромиссы между атрибутами качества (такими как производительность и безопасность) и компромиссы, необходимые для выполнения пользовательских, системных и бизнес-требований.