МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ МОЛДОВА

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ИНФОРМАТИКИ И МИКРОЭЛЕКТРОНИКИ

ДЕПАРТАМЕНТ ИНЖЕНЕРИИ СОФТВЭР И АВТОМАТИКИ

**Курсовая работа**

предмет «**Analiza si modulare sistemele informatice**»

Тема: База данных пропусков студентов

Выполнил: ст. гр. TI-154 Буянов Евгений

Проверил: ст.преподаватель Мельник Раду

Саченко Александр

Кишинев 2017

Оглавление

[Глава 1.1 Основные этапы развития UML 3](#_Toc500857900)

[Глава 2. Основные компоненты языка UML: 6](#_Toc500857901)

[Общая структура языка UML 6](#_Toc500857902)

[2.1 Диаграмма вариантов использования 13](#_Toc500857903)

[2.2 Диаграмма последовательности 20](#_Toc500857907)

[2.3 Диаграмма классов 25](#_Toc500857908)

[2.4 Диаграмма состояний. 27](#_Toc500857909)

[2.5 Диаграмма деятельности 30](#_Toc500857910)

[2.6 Диаграмма компонентов. 32](#_Toc500857911)

[2.7 Диаграмма развёртывания 35](#_Toc500857912)

Заключение……………………………………………………………………………………...37

# Глава 1.1 Основные этапы развития UML

Отдельные языки объектно-ориентированного моделирования стали появляться в период между серединой 1970-х и концом 1980-х годов, когда различные исследователи и программисты предлагали свои подходы к объектно-ориентированнному анализу и проектирования (ООАП). В период между 1989-1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50. Многие пользователи испытывали серьезные затруднения при выборе языка ООАП, поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем. Принятие отдельных методик и графических нотаций в качестве стандартов (IDEF0, IDEF1X) не смогло изменить сложившуюся ситуацию непримиримой конкуренции между ними в начале 90-х годов.

К середине 1990-х некоторые из методов были существенно улучшены и приобрели самостоятельное значение при решении различных задач ООАП. Наиболее известными в этот период становятся:

* Метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже - Booch'93).
* Метод Джеймса Румбаха (James Rumbaugh), получивший название Object Modeling Technique - ОМТ (позже - ОМТ-2).
* Метод Айвара Джекобсона (Ivar Jacobson), получивший название Object-Oriented Software Engineering - OOSE.

Каждый из этих методов был ориентирован на поддержку отдельных этапов ООАП. Например, метод OOSE содержал средства представления вариантов использования, которые имеют существенное значение на этапе анализа требований в процессе проектирования бизнес-приложений. Метод ОМТ-2 наиболее подходил для анализа процессов обработки данных в информационных системах. Метод Booch'93 нашел наибольшее применение на этапах проектирования и разработки различных программных систем.

История развития языка UML берет начало с октября 1994 года, когда Гради Буч и Джеймс Румбах из Rational Software Corporation начали работу по унификации методов Booch и ОМТ. Хотя сами по себе эти методы были достаточно популярны, совместная работа была направлена на изучение всех известных объектно-ориентированных методов с целью объединения их достоинств. Проект так называемого унифицированного метода (Unified Method) версии 0.8 был подготовлен и опубликован в октябре 1995 года. Осенью того же года к ним присоединился А. Джекобсон, главный технолог из компании Objectory AB (Швеция), с целью интеграции своего метода OOSE с двумя предыдущими.

Начиная работу по унификации своих методов, Г. Буч, Дж. Румбах и А. Джекобсон сформулировали следующие требования к языку моделирования. Он должен:

* Позволять моделировать не только программное обеспечение, но и более широкие классы систем и бизнес-приложений, с использованием объектно-ориентированных понятий.
* Явным образом обеспечивать взаимосвязь между базовыми понятиями для моделей концептуального и физического уровней.
* Обеспечивать масштабируемость моделей, что является важной особенностью сложных многоцелевых систем.
* Быть понятен аналитикам и программистам, а также должен поддерживаться специальными инструментальными средствами, реализованными на различных компьютерных платформах.

Разработка системы обозначений или нотации для ООАП оказалась непохожей на разработку нового языка программирования. Во-первых, необходимо было решить две проблемы:

1. Должна ли данная нотация включать в себя спецификацию требований?
2. Следует ли расширять данную нотацию до уровня языка визуального программирования?

Во-вторых, было необходимо найти удачный баланс между выразительностью и простотой языка. С одной стороны, слишком простая нотация ограничивает круг потенциальных проблем, которые могут быть решены с помощью соответствующей системы обозначений. С другой стороны, слишком сложная нотация создает дополнительные трудности для ее изучения и применения аналитиками и программистами. В случае унификации существующих методов необходимо учитывать интересы специалистов, которые уже имеют опыт работы с ними, поскольку внесение серьезных изменений в новую нотацию может повлечь за собой непонимание и неприятие ее пользователями прежних методик. Чтобы исключить неявное сопротивление со стороны отдельных специалистов, необходимо учитывать интересы самого широкого круга пользователей. Последующая работа над унифицированным языком моделирования (Unified Modeling Language, UML) должна была учесть все эти обстоятельства.

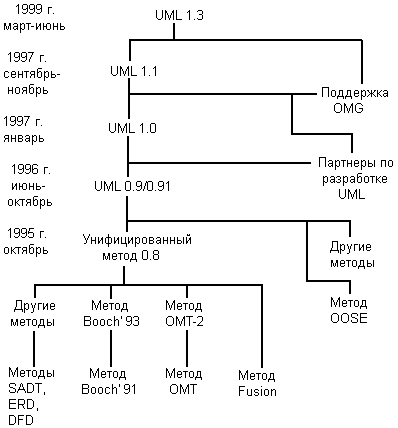
В этот период поддержка разработки языка UML становится одной из целей консорциума OMG (Object Management Group). Хотя консорциум OMG был образован еще в 1989 году с целью разработки предложений по стандартизации объектных и компонентных технологий CORBA, язык UML приобрел статус второго стратегического направления в работе OMG. Именно в рамках OMG создается команда разработчиков под руководством Ричарда Соли, которая будет обеспечивать дальнейшую работу по унификации и стандартизации языка UML.

Усилия Г. Буча, Дж. Румбаха и А. Джекобсона привели к появлению первых документов, содержащих описание собственно языка UML версии 0.9 (июнь 1996) и версии 0.91 (октябрь 1996). Имевшие статус запроса предложений RTP (Request For Proposals), эти документы послужили своеобразным катализатором для широкого обсуждения языка UML различными категориями специалистов. Первые отзывы и реакция на язык UML указывали на необходимость его дополнения отдельными понятиями и конструкциями.

Компания Rational Software вместе с несколькими организациями, изъявившими желание выделить ресурсы для разработки строгого определения версии 1.0 языка UML, учредила консорциум партнеров UML, в который первоначально вошли такие компании, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI и Unisys. Эти компании обеспечили поддержку последующей работы по более точному и строгому определению нотации, что привело к появлению версии 1.0 языка UML.

Специфика языка UML заключается в том, что он определяет семантическую метамодель, а не модель конкретного интерфейса и способы представления или реализации компонентов.

В настоящее время все вопросы дальнейшей разработки языка UML сконцентрированы в рамках консорциума OMG. Соответствующая группа специалистов обеспечивает публикацию материалов, содержащих описание последующих версий языка UML и запросов предложений RFP по его стандартизации. Очередной этап развития данного языка закончился в марте 2004 года, когда консорциумом OMG было опубликовано описание языка UML 2.0. История разработки и последующего развития языка UML графически представлена на рис. 1.1.



**Рис. 1.1.** История развития языка UML

На основе технологии UML Microsoft, Rational Software и другие поставщики средств разработки программных систем разработали единую информационную модель, которая получила название UML Information Model. Предполагается, что эта модель даст возможность различным программам, поддерживающим идеологию UML, обмениваться между собой компонентами и описаниями. Все это позволит создать стандартный интерфейс между средствами разработки приложений и средствами визуального моделирования.

Уже в настоящее время разработаны средства визуального программирования на основе UML, обеспечивающие интеграцию, включая прямую и обратную генерацию кода программ, с наиболее распространенными языками и средами программирования, такими как MS Visual C++, Java, C#, Object Pascal/Delphi, Power Builder, MS Visual Basic, Forte, Ada, Smalltalk. Поскольку при разработке языка UML были приняты во внимание многие передовые идеи и методы, можно ожидать, что на очередные версии языка UML также окажут влияние и другие перспективные технологии и концепции. Кроме того, на основе языка UML могут быть определены многие новые перспективные методы. Язык UML может быть расширен без переопределения его ядра.

# Глава 2. Основные компоненты языка UML:

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения. Этот язык вобрал в себя наилучшие качества методов программной инженерии, которые с успехом использовались на протяжении последних лет при моделировании больших и сложных систем.

Язык UML основан на некотором числе базовых понятий, которые могут быть изучены и применены большинством программистов и разработчиков, знакомых с методами объектно-ориентированного анализа и проектирования. При этом базовые понятия могут комбинироваться и расширяться таким образом, что специалисты объектного моделирования получают возможность самостоятельно разрабатывать модели больших и сложных систем в самых различных областях приложений.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи, которые могут быть решены с использованием данных моделей. При этом одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

## Общая структура языка UML

UML состоит из двух взаимодействующих частей, таких как:

* Семантика языка UML. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML.
* Нотация языка UML. Представляет собой графическую нотацию для визуального представления семантики языка UML.

Абстрактный синтаксис и семантика языка UML описываются с использованием некоторого подмножества нотации UML. В дополнение к этому, нотация UML описывает соответствие или отображение графической нотации в базовые понятия семантики. Таким образом, с функциональной точки зрения эти две части дополняют друг друга. При этом семантика языка UML описывается на основе некоторой метамодели, имеющей три отдельных представления: абстрактный синтаксис, правила корректного построения выражений и семантику. Рассмотрение семантики языка UML предполагает некоторый "полуформальный" стиль изложения, который объединяет естественный и формальный языки для представления базовых понятий и правил их расширения.

Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения. Структурные модели, известные также как статические модели, описывают структуру сущностей или компонентов некоторой системы, включая их классы, интерфейсы, атрибуты и отношения. Модели поведения, называемые иногда динамическими моделями, описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Для решения столь широкого диапазона задач моделирования разработана достаточно полная семантика для всех компонентов графической нотации. Требования семантики языка UML конкретизируются при построении отдельных видов диаграмм. Нотация языка UML включает в себя описание отдельных семантических элементов, которые могут применяться при построении диаграмм.

Формальное описание самого языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

* Мета-метамодель
* Метамодель
* Модель
* Объекты пользователя

Уровень мета-метамодели образует исходную основу для всех метамодельных представлений. Главное предназначение этого уровня состоит в том, чтобы определить язык для спецификации метамодели. Мета-метамодель определяет модель языка UML на самом высоком уровне абстракции и является наиболее компактным ее описанием. С другой стороны, мета-метамодель может специфицировать несколько метамоделей, чем достигается потенциальная гибкость включения дополнительных понятий. Примерами понятий этого уровня служат метакласс, метаатрибут, метаоперация. Семантика мета-метамодели не входит в описание языка UML. Это делает язык UML более простым для изучения, поскольку не требует знания общей теории формальных языков и формальной логики.

Метамодель является экземпляром или конкретизацией мета-метамодели. Главная задача этого уровня - определить язык для спецификации моделей.

Описание семантики языка UML предполагает рассмотрение базовых понятий только уровня метамодели, который представляет собой лишь пример или частный случай уровня мета-метамодели.

Специфика языка UML

UML является стандартным инструментом для создания документированных каркасов ("чертежей") программного обеспечения. UML - это язык для визуализации, специфицирования, конструирования и документирования артефактов программных систем. Напомним что, артефакт (artifact) - диаграмма, документ, модель, закон и т. д. - нечто, описывающее определенное понятие предметной области. UML разработан таким образом, чтобы удовлетворять потребности при моделировании любых систем: от информационных систем масштаба предприятия до распределенных Web-приложений и даже встроенных систем реального времени. Это выразительный язык, позволяющий рассмотреть систему со всех точек зрения, имеющих отношение к ее разработке и последующему развертыванию. Несмотря на обилие выразительных возможностей, этот язык прост для понимания и использования.

**Сущности UML**

В UML имеется четыре типа сущностей:

* структурные;
* поведенческие;
* группирующие;
* аннотационные.

Сущности являются основными объектно-ориентированными элементами языка. С их помощью можно создавать корректные модели. Структурные сущности - это имена существительные в моделях на языке UML. Как правило, они представляют собой статические части модели, соответствующие концептуальным или физическим элементам системы. Существует семь разновидностей структурных сущностей.

*Класс* (class) - это описание совокупности объектов с общими атрибутами, операциями отношениями и семантикой. Графически класс изображается в виде прямоугольника, в котором записаны его имя, атрибуты и операции, например как это показано на рис. 2.2**.**:

|  |
| --- |
| *ClassName* |
| -PrivateAttribute : char #ProtectedAttribute +PublicAttribute |
| +Operation1(S : String) +Operation2() |

**Рис. 2.2.** Пиктограмма класса

*Интерфейс* (interface) - это совокупность операций, которые определяют определенную службу (сервис, набор услуг), которые предоставляет класс или компонент. На диаграммах интерфейс изображается в виде круга, под которым указывается его имя, как это показано на рис. 2.3. Интерфейс очень редко, практически никогда, существует сам по себе - обычно он присоединяется к реализующему его классу или компоненту.

uml_interface

**Рис. 2.3.**  Пиктограмма интерфейса

*Кооперация* (collaboration) определяет взаимодействие, она представляет собой совокупность ролей и других элементов, которые, работая вместе, производят некоторый кооперативный эффект, не сводящийся к обычно сумме слагаемых. Графически кооперация изображается в виде эллипса, который ограничивается пунктиром, внутри обычно заключено только имя, пример на рис. 2.4.

uml_collaboration

**Рис. 2.4.**  Пиктограмма кооперации

*Прецедент* (use case) - это описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-то определенного актера (actor). Графически прецедент тоже изображается в виде эллипса, только ограниченного непрерывной линией, обычно содержащего только его имя, как показано на рис. 2.5.

uml_usecase

**Рис. 2.5.** Пиктограмма прецедента

*Активным классом* (active class) называется класс, объекты которого вовлечены в один или несколько процессов, или нитей (threads), и поэтому могут инициировать управляющее воздействие. Графически активный класс изображается также как и простой класс, но ограничивается прямоугольником, который рисуется жирной линией, и включает имя, атрибуты и операции, пример на рис. 2.6.

|  |
| --- |
| *ClassName* |
| -PrivateAttribute:char #ProtectedAttribute +PublicAttribute |
| +Operation1(int S:String)  +Operation2() |

**Рис. 2.6.** Пиктограмма активного класса

*Компонент* (component) - это физическая заменяемая часть системы, которая соответствует некоторому набору интерфейсов и обеспечивает его реализацию. Графически компонент изображается в виде прямоугольника с вкладками, содержащего обычно только имя, как показано на рис. 2.7.

uml_component

**Рис. 2.7.** Пиктограмма компонента

*Узел* (node) - это элемент реальной (физической) системы, который существует во время функционирования программного продукта и представляет собой некоторый вычислительный ресурс, обычно обладающий как минимум некоторым объемом памяти, а часто еще и возможностью обработки. Графически для изображения узла используется куб, обычно содержащий только имя узла, пример на рис. 2.8.

uml_node

**Рис. 2.8.** Пиктограмма узла

Перечисленные семь базовых элементов: классы, интерфейсы, кооперации, прецеденты, активные классы, компоненты и узлы - являются основными структурными сущностями, которые могут быть использованы в модели UML. Существуют и другие разновидности структурных сущностей: актеры, сигналы, утилиты (виды классов), процессы и нити (виды активных классов), приложения, документы, файлы, библиотеки, страницы и таблицы (виды компонентов).

Поведенческие сущности (behavioral things) являются динамическими составляющими модели UML. Это глаголы языка, они описывают поведение модели во времени и в пространстве. Существует всего два основных типа поведенческих сущностей.

*Взаимодействие* (interaction) - это поведение, суть которого заключается в обмене сообщениями (messages) между объектами в рамках конкретного контекста для достижения определенной цели. С помощью взаимодействия можно описать как отдельную операцию, так и поведение совокупности объектов. Взаимодействие предполагает ряд других элементов, таких как сообщения, последовательности действий (поведение, инициированное сообщениями) и связи (между объектами). Графически сообщение изображается в виде стрелки. Над которой почти всегда пишется имя соответствующей операции, как показано на рис. 2.9.

uml_interaction

**Рис. 2.9.**  Пиктограмма сообщения

*Автомат* (state machine) - алгоритм поведения, определяющий последовательность состояний, через которые объект или взаимодействие проходят на протяжении своего жизненного цикла в ответ на различные события, а также реакции на эти события. С помощью автоматов описываются поведение отдельного класса или кооперации классов. С автоматом связан ряд других элементов: состояния, переходы из одного состояния в другое, события - сущности инициирующие переходы и виды действий - реакция на переходы. Графически состояние изображается в виде прямоугольника с закругленными углами, содержащего имя и, возможно, промежуточные состояния, пример на рис. 2.10.

uml_state

**Рис. 2.10.**  Пиктограмма состояния

Взаимодействия и автоматы являются основными поведенческими сущностями, входящими в модель UML. Семантически они часто бывают связаны с различными структурными элементами, в первую очередь с классами, кооперациями и объектами.

Группирующие сущности являются организующими частями модели UML. Это блоки, на которые можно разложить модель. Такая первичная сущность имеется в единственном экземпляре - это пакет.

*Пакеты* (packages) представляют собой универсальный механизм организации элементов в группы. В пакет можно поместить структурные, поведенческие и другие группирующие сущности. В отличие от компонентов, которые реально существуют во время работы программы, пакеты носят чисто концептуальный характер, то есть существуют только в процессе разработки. Для изображения пакета используется пиктограмма папки с закладкой, содержащей обычно только имя, но иногда и содержимое, см. рис. 2.11.



**Рис. 2.11.** Пиктограмма пакета

Аннотационные сущности - пояснительные части модели UML. Это комментарии для дополнительного описания, разъяснения или замечания к любому элементу модели. Имеется только один базовый тип аннотационных элементов - примечание.

*Примечание* (note) - это просто символ для изображения комментариев или ограничений, присоединенный к элементу или группе элементов. Графически примечание изображается в виде прямоугольника с загнутым краем, содержащим текстовый или графический комментарий, как показано на рис. 2.12.

uml_note

**Рис. 2.12.**  Пиктограмма примечания

Этот элемент является основной аннотационной сущностью, которую мы можем использовать в модели UML. Чаще всего примечания используют, чтобы снабдить диаграммы комментариями или ограничениями, которые можно выразить в виде неформального или формального текста. Существуют варианты этого элемента, например требования, в которых описывают некоторое желательное поведение системы с точки зрения внешней по отношению к модели.

**Отношения UML**

В языке UML определены четыре типа отношений:

* Отношение зависимости (dependency relationship)
* Отношение ассоциации (association relationship)
* Отношение обобщения (generalization relationship)
* Отношение реализации (realization relationship)

Эти отношения являются основными связующими конструкциями в UML и применяются для построения корректных моделей.

*Зависимость* (dependency) - это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой. Графически для изображения зависимости используют пунктирную линию, обычно со стрелкой, которая может содержать метку (см. рис. 2.13).

uml_dependency

**Рис. 2.13.**  Пиктограмма зависимости

*Ассоциация* (association) - структурное отношение, описывающее совокупность связей, где под связью понимается некоторая смысловая связь между объектами. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации. Графически ассоциация изображается в виде линии (иногда завершающейся стрелкой или содержащей метку). На рис. 2.14. показан пример отношений этого вида.

uml_association

**Рис. 2.14.**  Пиктограмма ассоциации

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму - отношение композиции.

*Агрегирование* (aggregation) - так называется структурное отношение между целым и его частями. Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа "часть-целое". Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой. С точки зрения модели отдельные части системы могут выступать как в виде элементов, так и в виде подсистем, которые, в свою очередь, тоже могут образовывать составные компоненты или подсистемы. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на "целое" остальные являются его "частями" (рис. 2.15).

gl5-8

**Рис. 2.15.** Пиктограмма агрегации

Отношение *композиции* является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на класс-композицию или "целое" остальные являются его "частями" (рис. 2.16).

gl5-10

**Рис. 2.16.** Пиктограмма композиции

*Обобщение* (generalization) - это отношение "специализация/обобщение", при котором объект специализированного элемента (проще говоря, потомок) может быть подставлен вместо объекта обобщенного элемента (родителя, предка). Как и положено в объектно-ориентированном программировании, потомок (child) наследует структуру и поведение своего предка (parent). Графически отношение обобщения изображается в виде линии с незакрашенной стрелкой, указывающей на предка. Как показано на рис. 2.17.

uml_generalization

**Рис. 2.17.**  Пиктограмма обобщения

*Реализация* (realization) - это семантическое отношение между классификаторами, при котором один классификатор определяет обязательство, а другой гарантирует его выполнение. Отношение реализации встречаются в двух случаях: во-первых, между интерфейсами и реализующими их классами или компонентами, а во-вторых, между прецедентами и реализующими их кооперациями. Отношение реализации изображается в виде пунктирной линии с незакрашенной стрелкой, как нечто среднее между отношениями обобщения и зависимости (см. рис. 2.18).

uml_realization

**Рис. 2.18.**  Пиктограмма реализации

Диаграммы UML:

1. Диаграмма вариантов использования (use case diagram)
2. Диаграмма взаимодействий (interaction diagram)

* Диаграмма последовательности (sequence diagram)
* Диаграмма коопераций (collaboration diagram)

1. Диаграмма классов (class diagram)
2. Диаграмма состояний (statechart diagram)
3. Диаграмма деятельности (activity diagram)
4. Диаграмма реализации (implementation diagrams)

* Диаграмма компонентов (component diagram)
* Диаграмма развертывания (deployment diagram)

## 2.1 Диаграмма вариантов использования

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

В самом общем случае, диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами. При этом отдельные компоненты диаграммы могут быть заключены в прямоугольник, который обозначает проектируемую систему в целом.

Каждый **вариант использования** определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

http://ooad.asf.ru/student/lectures/risp/images/uml_usecase.gif

**Актер** представляет собой любую внешнюю по отношению к моделируемой

системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой.

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса.

gl4-2

В языке UML определены четыре типа отношений:

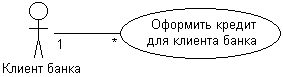
* Отношение зависимости (dependency relationship)
* Отношение ассоциации (association relationship)
* Отношение обобщения (generalization relationship)
* Отношение реализации (realization relationship)

Эти отношения являются основными связующими конструкциями в UML и применяются для построения корректных моделей.

### Отношение ас социации

Отношение ассоциации является одним из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования



**Рис.** Пример графического представления отношения ассоциации между актером

и вариантом использования

### Отношение расширения

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом "extend" ("расширяет"), как показано на рис ниже.

http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl4/gl4-7.jpg

**Рис.** Пример графического изображения отношения расширения между

вариантами использования

### Отношение включения

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl4/gl4-11.jpg

**Рис.** Пример графического изображения отношения включения между вариантами использования

**1. Варианты использования для основной страницы.**

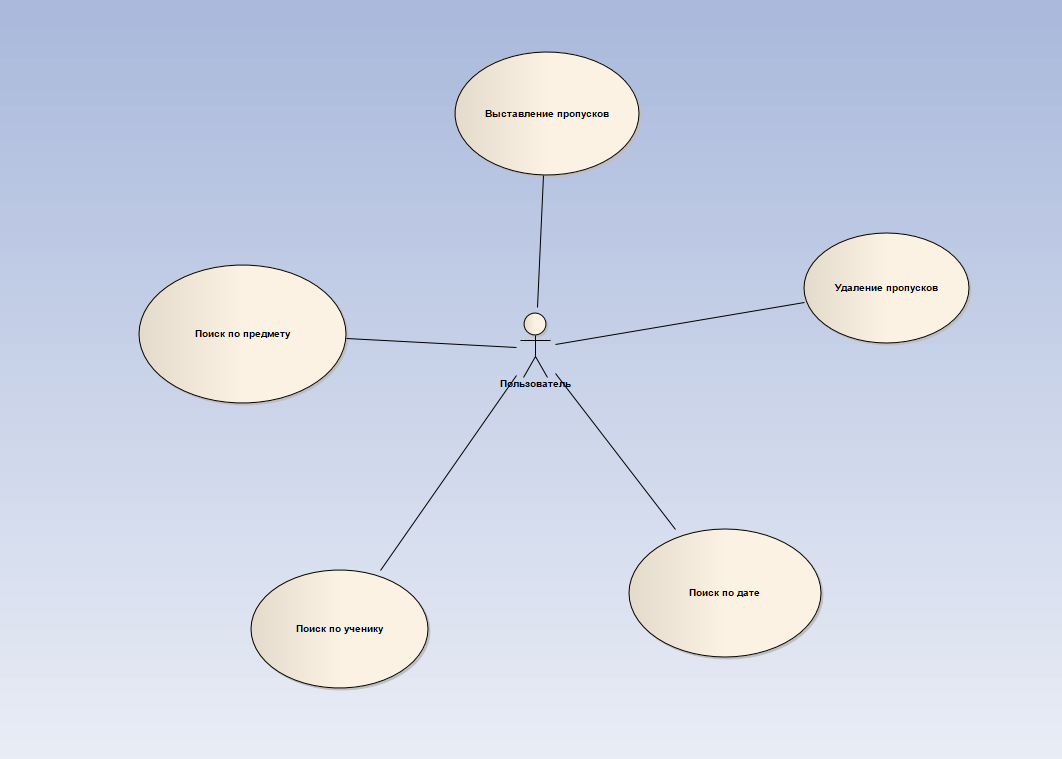


Диаграмма 1 – «Диаграмма вариантов использования для основной страницы»

«База данных проппусков студентов»  
   
В диаграмме 1 , мы проектируем общую структура нашего музыкального приложения.  
У пользователю доступны 5 основных функции :  
1.Выставление пропусков  
2.Удаление пропусков  
3.Поиск по дате  
4.Поиск по предмету  
5.Поиску по ученику

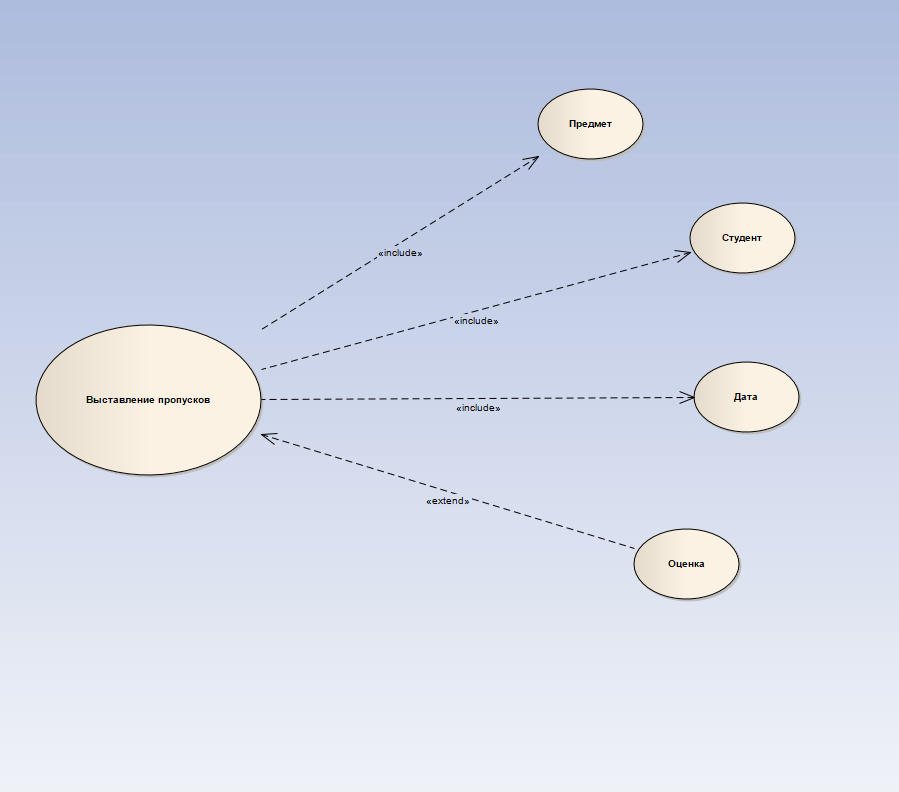
**2.Диаграмма для “Выставление пропусков”**  
  


Диаграмма 2 – «Диаграмма вариантов использования для выставления пропусков

Для диаграммы «Выставление пропусков» обязательными является: Предмет, Студент и Дата. Однако у пользователя также есть возможность выставить оценку для данной записи в бд.

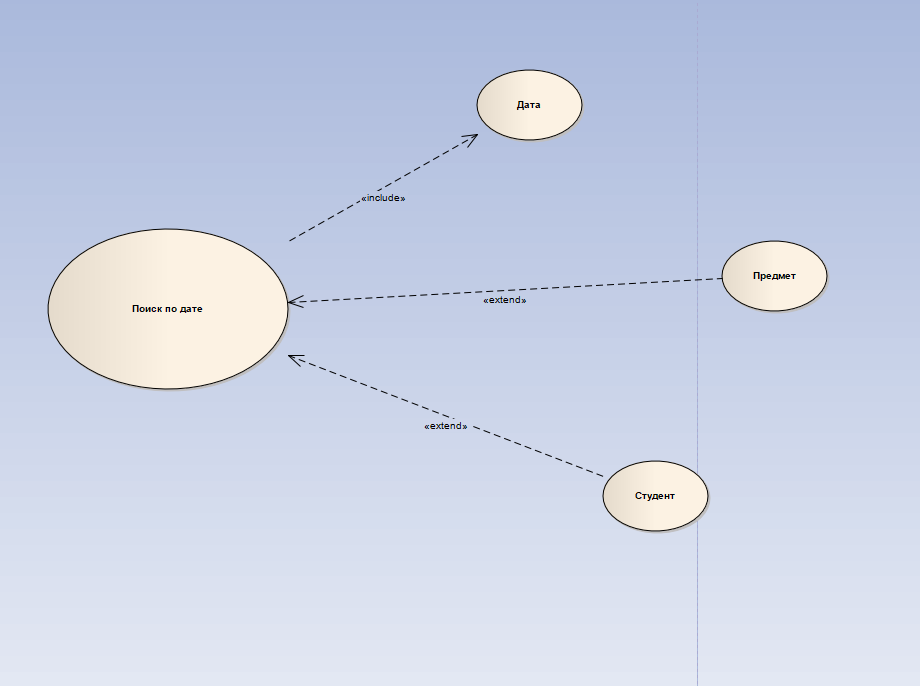
**3.Диаграмма для “ Поиск по дате ”**

Диаграмма 3 – «Диаграмма вариантов использования для поиска по дате»

Для диаграммы «Поиск по дате» обязательными является: Дата. Для более детального поиска существуют необязательные поля «Предмет» и «Студент». При случае когда не выбраны необязательные поля выводится список из всех студентов и всех предметов за выбранную дату.

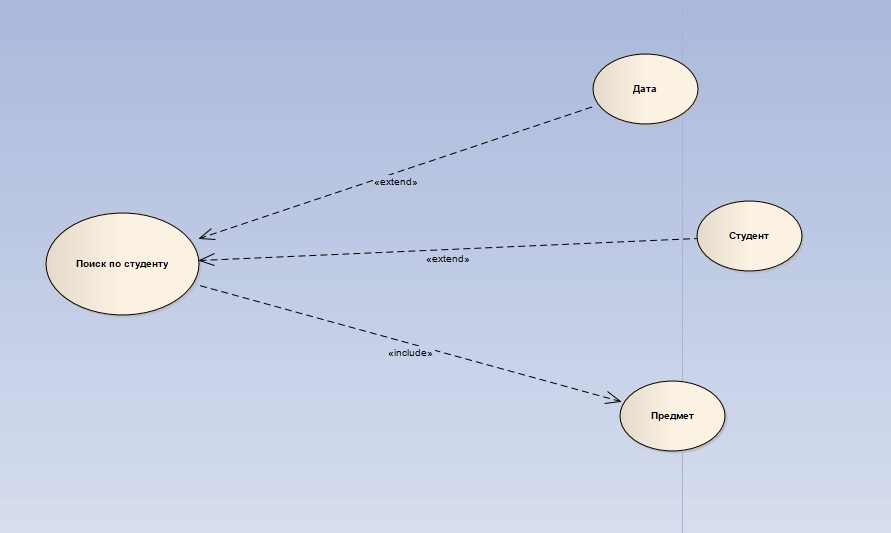
**4.Диаграмма для “ Поиск по студенту ”**

Диаграмма 4 – «Диаграмма вариантов использования для поиска по студенту»

Для диаграммы «Поиск по студенту» обязательными является: Студент. Для более детального поиска существуют необязательные поля «Предмет» и «Дата». При случае когда не выбраны необязательные поля выводится список из всех предметов за все время по выбранному студенту.

2.2 Диаграмма последовательности

– диаграмма, которая описывает временные особенности передачи и приема сообщений между объектами.

Диаграмма кооперации используется для представления структурных особенностей передачи и приема сообщений между объектами.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control). Сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому.

В языке UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указаны на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому они относятся. В этом случае они записываются в кавычках. Используются следующие обозначения для моделирования действий:

 "call" (вызвать) - сообщение, требующее вызова операции или процедуры принимающего объекта;  "return" (возвратить) - сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту;  "create" (создать) - сообщение, требующее создания другого объекта для выполнения определенных действий;

 "destroy" (уничтожить) - сообщение с явным требованием уничтожить соответствующий объект. Посылается в том случае, когда необходимо прекратить нежелательные действия со стороны существующего в системе объекта, либо, когда объект больше не нужен и должен освободить задействованные им системные ресурсы;

"send" (послать) - обозначает посылку другому объекту некоторого сигнала, который асинхронно инициируется одним объектом и принимается (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Для графического изображения объектов на диаграмме кооперации используется такой же символ прямоугольника, что и для классов.

Понятие кооперации (collaboration) служит для обозначения множества взаимодействующих с определенной целью объектов в общем контексте моделируемой

системы. Цель самой кооперации состоит в том, чтобы специфицировать особенности

реализации отдельных наиболее значимых операций в системе.

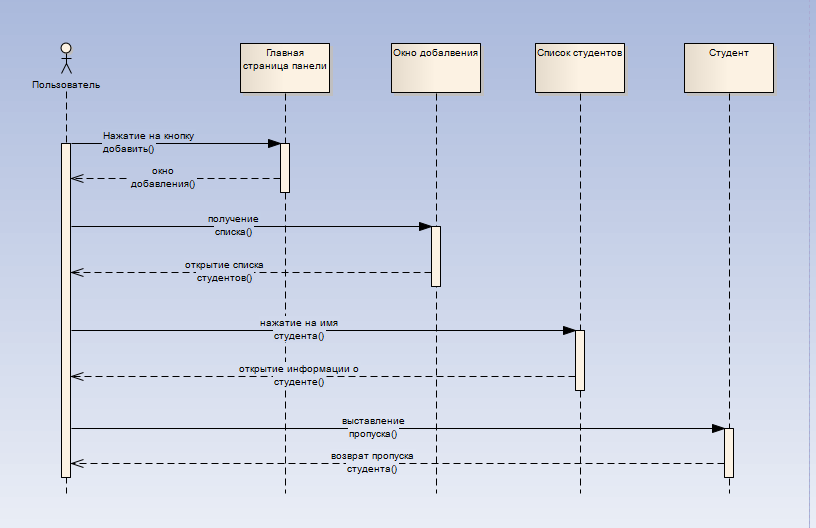


Диаграмма 5 – «Диаграмма последовательности выставление пропусков»

Данная диаграмма последовательностей объясняет действия пользователя, чтобы выставить пропуск студенту. Для того чтобы выполнить данное действие необходимо перейти на «Страницу выставления пропуска», выбрать студента и предмет.

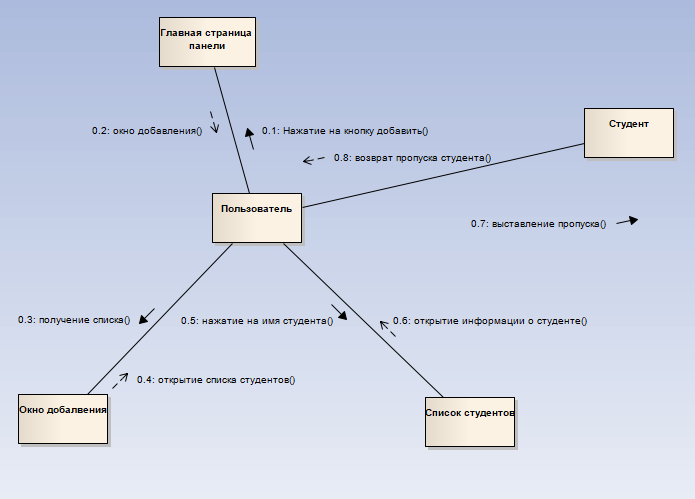


Диаграмма 6 – «Диаграмма коммуникации выставления пропусков»

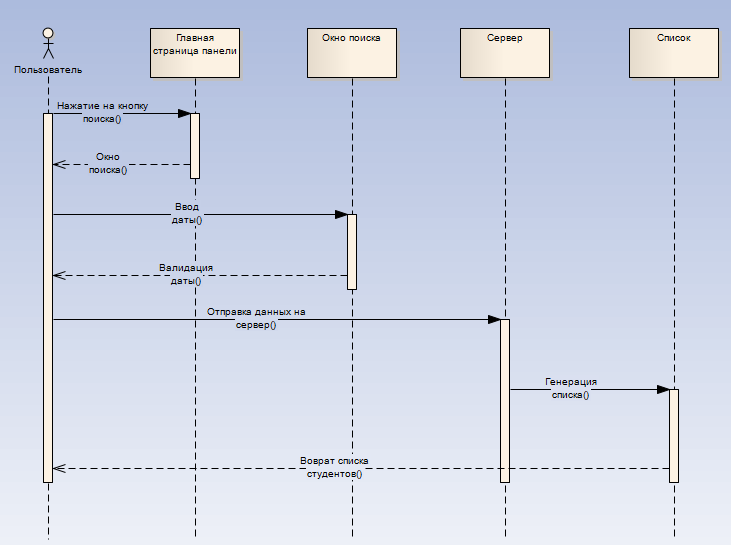


Диаграмма 7 – «Диаграмма последовательности поиска студента по дате»

Данная диаграмма последовательностей показывает, какими сообщениями обменивается пользователь базы данных чтобы выполнить поиск студента по дате.

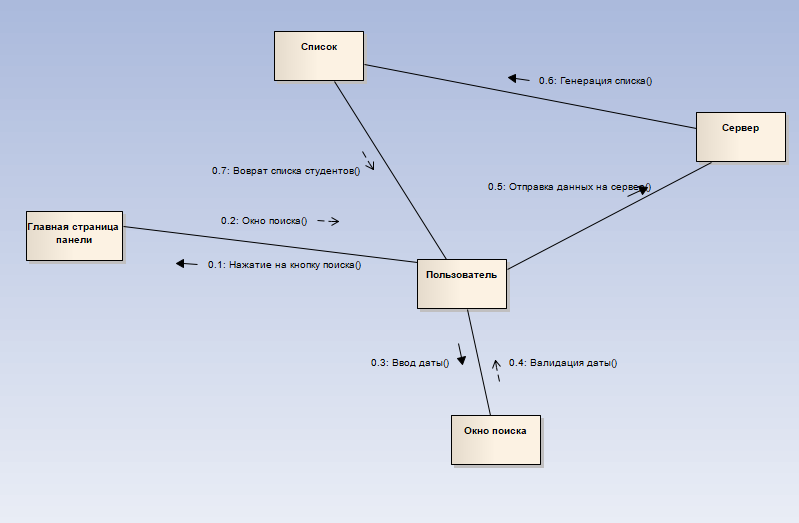


Диаграмма 8 – «Диаграмма коммуникации поиска студента по дате»

Диаграмма кооперации для «Удаление пропуска»

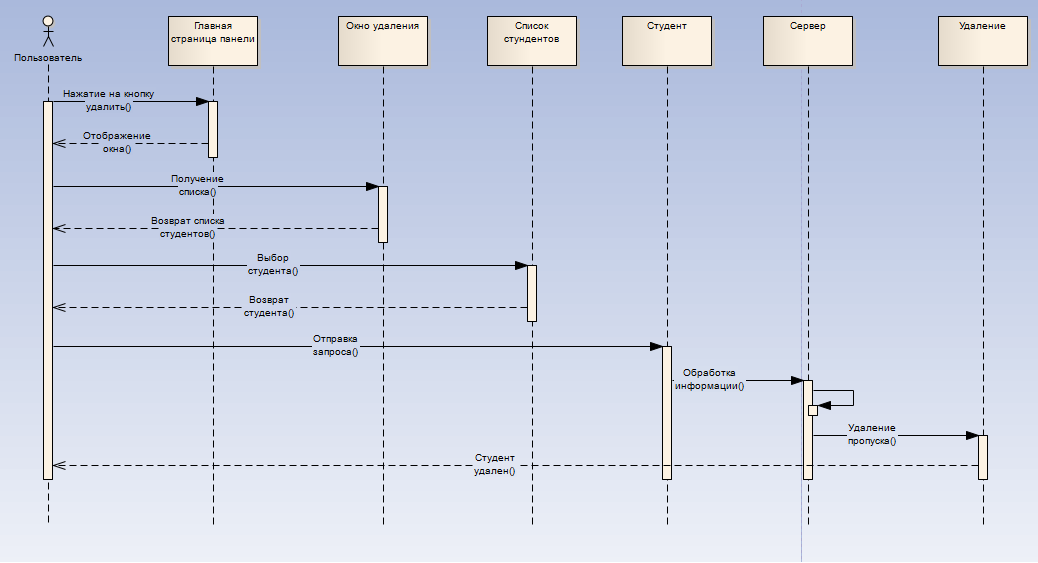


Диаграмма 9 – «Диаграмма последовательности удаления пропуска»

Данная диаграмма последовательностей показывает, какими сообщениями обменивается пользователь для удаления пропуска выставленного студенту.

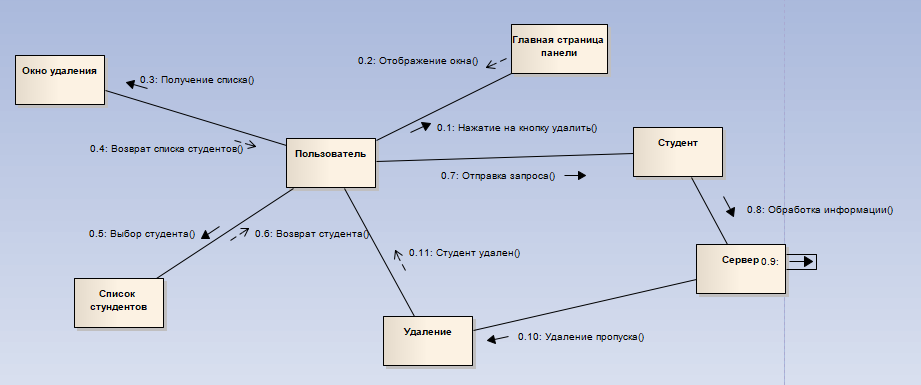


Диаграмма 10 – «Диаграмма коммуникации удаления пропуска»

## 2.3 Диаграмма классов

**Теоретические сведения**

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Базовыми отношениями или связями в языке UML являются:

* Отношение зависимости (dependency relationship)
* Отношение ассоциации (association relationship)
* Отношение обобщения (generalization relationship) -
* Отношение реализации (realization relationship)

Интерфейсы являются элементами диаграммы вариантов использования. Однако при построении диаграммы классов отдельные интерфейсы могут уточняться и в этом случае для их изображения используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом "interface".

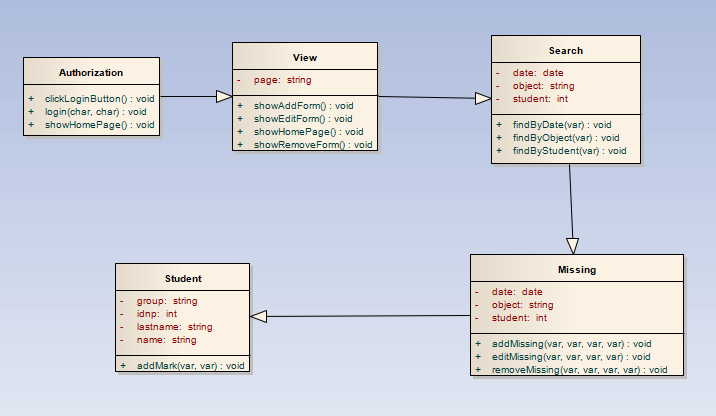


Диаграмма 11 – «Диаграмма классов»

В данной диаграмме описываются классы, используемые для реализации «Базы данных пропусков студентов».

* Класс Authorization отвечает за авторизацию пользователя и защиту веб приложения.
* Класс View отвечает за вывод контента и оформление на странице
* Класс Search отвечает за поиск данных в базе
* Класс Student отвечает за каждого отдельного студента, пропуск которого выставляется в базе
* Класс Missing отвечает за данные передаваемые базе для выставления пропуска

## 2.4 Диаграмма состояний

**Теоретические сведения**

Диаграмма состояний – это диаграмма, главное предназначение которой - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Состояние (state) - абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим.

Событие (event), формально, представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели. В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение. Введение для перехода сторожевого условия позволяет явно специфицировать семантику его срабатывания.

Выражение действия (action expression) выполняется в том и только в том случае, когда переход срабатывает. Представляет собой атомарную операцию (достаточно простое вычисление), выполняемую сразу после срабатывания соответствующего перехода до начала каких бы то ни было действий в целевом состоянии.

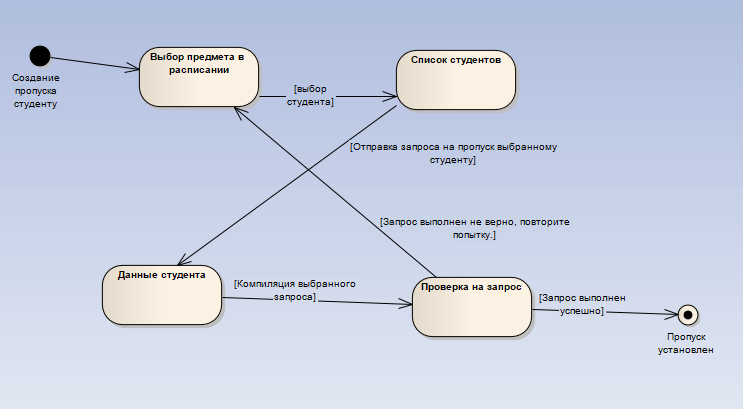


Диаграмма 12 – «Диаграмма состояний выставления пропуска студенту »

В данной диаграмме отражены состояния которые может получить пользователь при выставлении пропуска. В диаграмме описан процесс выставления пропуска в котором учувствуют «Предмет», «Студент», «Валидация на сервере», «Добавление пропуска в базу данных».

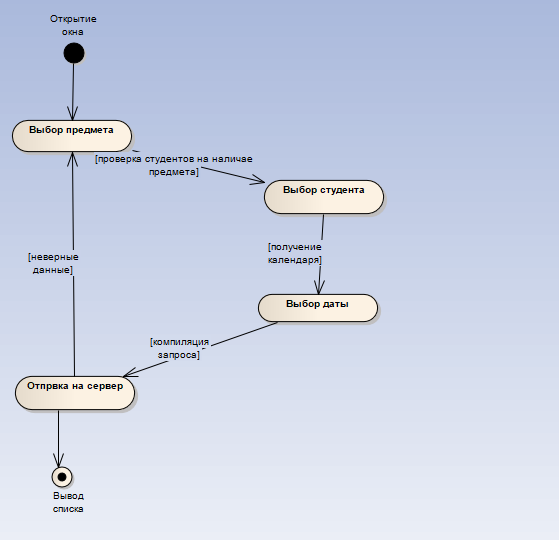


Диаграмма 13 – «Диаграмма состояний поиска»

В диаграмме описывается процесс поиска предмета с использованием дополнительных параметров, таких как: дата, студент.

## 2.5 Диаграмма деятельности

**Теоретические сведения**

Диаграмма деятельности – это диаграмма, главное предназначение которой - моделирования процесса выполнения операций. Отличие от диаграммы состояний заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии.

Состояние действия (action state) является специальным случаем состояния. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим.

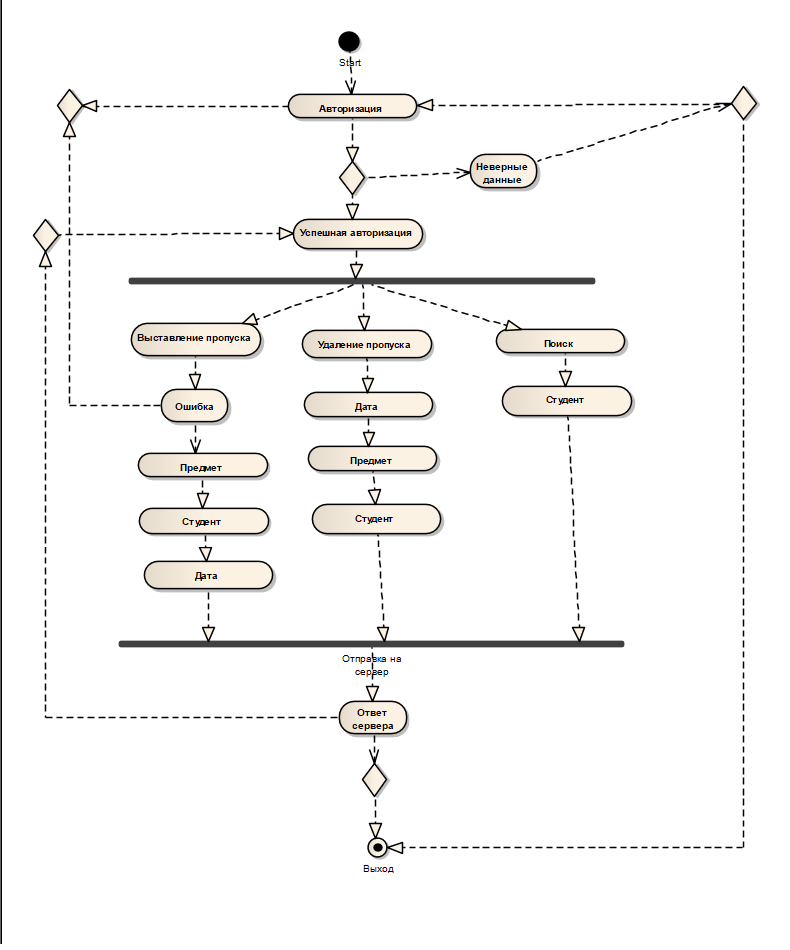


Диаграмма 14 – «Диаграмма деятельности»

Диаграмма описывает все этапы пользователя для выставления пропуска студенту или поиска студента, удаления пропуска . В элементы безопасности входят сторожевые условия при неудачной авторизации и при ошибки в результате обработки запроса на сервере.

## 2.6 Диаграмма компонентов.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. .Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

* Визуализации общей структуры исходного кода программной системы.
* Спецификации исполнимого варианта программной системы.
* Обеспечения многократного использования отдельных фрагментов программного кода.
* Представления концептуальной и физической схем баз данных.

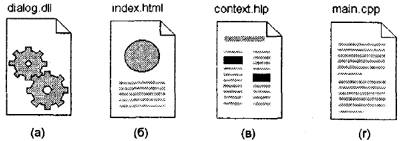
В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие - на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве классификаторов.

Виды компонентов

Поскольку компонент как элемент физической реализации модели представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации. Строго.говоря, эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления. Некоторые из таких общепринятых обозначений для компонентов изображены ниже (рис. 2).

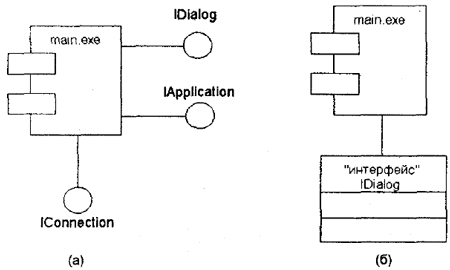
В языке UML выделяют три вида компонентов.

* Во-первых, компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 9.2, а), Web-страницы на языке разметки гипертекста с расширением html (рис. 9.2, б) и файлы справки с расширением htр (рис. 9.2, в).
* Во-вторых, компоненты-рабочие продукты. Как правило - это файлы с исходными текстами программ, например, с расширениями h или срр для языка C++ (рис. 9.2, г).
* В-третьих, компоненты исполнения, представляющие исполнимые модули - файлы с расширением ехе. Они обозначаются обычным образом.

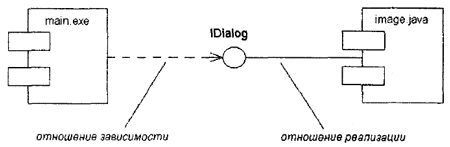


**Интерфейсы**

Следующим элементом диаграммы компонентов являются интерфейсы. Напомним, что в общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 9.3, а). При этом имя интерфейса, которое обязательно должно начинаться с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.



**Зависимости**

Зависимости могут отражать связи модулей программы на этапе компиляции и генерации объектного кода. В другом случае зависимость может отражать наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой. 

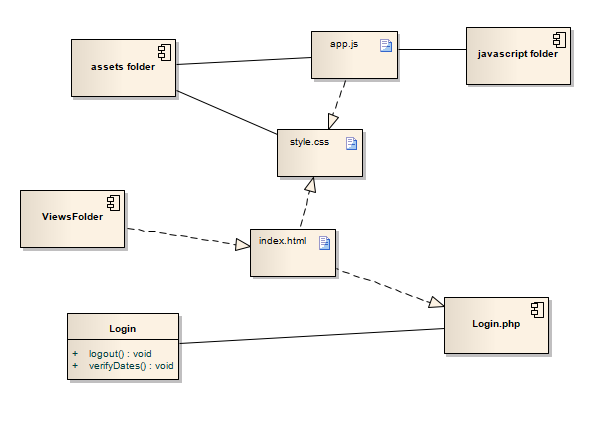


Диаграмма 15 – «Диаграмма компонентов страницы автоизации»

Данная диаграмма описывает внешний вид главной страницы, все элементы которые используются для визуализации и получения данных при авторизации. Индексная страница при авторизации будет index.html использующая элементы папки assets (style.css), javascript folder (app.js). При авторизации используется класс Login который находится в файле Login.php.

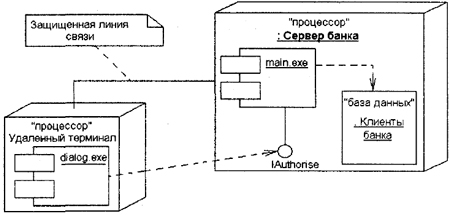
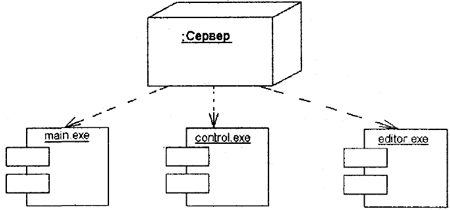
## 2.7 Диаграмма развёртывания

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Конечно, если разрабатывается простая программа, которая может выполняться локально на компьютере пользователя, не задействуя никаких периферийных устройств и ресурсов, то в этом случае нет необходимости в разработке дополнительных диаграмм. Однако при разработке корпоративных приложений ситуация представляется совсем по-другому.

Во-первых, сложные программные системы могут реализовываться в сетевом варианте на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач по рациональному размещению компонентов по узлам этой сети, что определяет общую производительность программной системы.

Во-вторых, интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности, криптозащищенности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

Наконец, технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. Эти аспекты также требуют визуального представления с целью спецификации программных и технологических особенностей реализации распределенных архитектур.



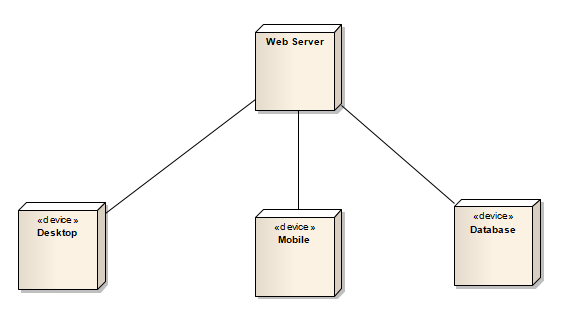


Диаграмма 16 – «Диаграмма развертывания базы данных»

На данной диаграмме выполнена дивизация устройств на которых возможно использование базы данных.

## Заключение:

Выполнив данную работу были использованы все теоретические знания, полученные во время посещения лекций. В работе использовались базовые понятия и знания языка UML, которые представлены в виде скриншотов из выполненных работ. При выполнении диаграмм «Базы данных пропусков студентов», были выполнены диаграммы «Диаграмма вариантов использования», «Диаграмма компонентов», «Диаграмма последовательности», «Диаграмма классов», «Диаграмма деятельности», «Диаграмма развертывания». Данная курсовая работа представляет собой отчет о проделанной деятельности за время обучения по данному курсу.

В функционал веб приложения входит:

* Авторизация
* Выставление пропуска студенту
* Удаление пропуска
* Выставление оценки (опционально)
* Поиск
  + По предмету
  + По студенту
  + По дате

Данный функционал представлен в «Диаграмме вариантов использования», которая описывает возможности пользователя формы добавления контента. В «Диаграмме последовательности» мы описали как происходит выставление пропуска с точки зрения структуры а так же алгоритма, тем самым задействовав обработку запроса на сервере. В «Диаграмме классов» были отображены все классы а так же методы необходимые для реализации данного приложения, включая взаимосвязь между всеми классами. Диаграмма состояний описала состояния которые могут возникнуть при выставлении оценки или поиске предмета, таким образом был тоже задействован алгоритм и структура приложения. В «Диаграмме деятельности» была создана модель пользователя использующего нашу базу, таким образом мы описали все возможные действия и результаты которые он может получить в ходе работы в приложении.