Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

И. Н. ТОНКОВИЧ, А. В. ШЕЛЕСТ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: ПРОЕКТИРОВАНИЕ, КОНСТРУИРОВАНИЕ И ВНЕДРЕНИЕ

Методическое пособие
по дисциплине «Технологии проектирования
сложных информационных систем»
для студентов очной формы обучения учреждений высшего
образования направления специальности
1-40 05 01-10 Информационные системы и технологии
(в бизнес-менеджменте)

Минск БГУИР 2023

УДК 004.414.38
ББК 16.332
Т 57

Рецензенты:
кафедра «Программное обеспечение информационных систем и технологий» Белорусского национального технического университета (заведующий кафедрой кандидат технических наук, доцент Ю.В. Полозков);
А. Н. Лавренов, доцент кафедры информатики и методики преподавания информатики учреждения образования «Белорусский государственный педагогический университет имени Максима Танка», кандидат физико-математических наук, доцент

Тонкович, И. Н.

Т 57 **Разработка программного обеспечения: проектирование, конструирование и внедрение**: метод. пособие по дисциплине «Технологии проектирования сложных информационных систем» для студентов очной формы обучения учреждений высшего образования направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте) / И. Н. Тонкович, А. В. Шелест. – Репозиторий БГУИР, 2023. – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/51273.

ISSN 2410-4655

В пособии рассматриваются краткие теоретические сведения и методические указания по выполнению лабораторных работ (разработка программного обеспечения: проектирование, конструирование и внедрение) по дисциплине «Технологии проектирования сложных информационных систем». Представленный материал предназначен для студентов очной формы обучения направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте). Методические указания по выполнению лабораторных работ, теоретический материал могут быть использованы студентами заочной и дистанционной форм обучения, а также магистрантами и специалистами при решении практических задач в области проектирования и разработки программного обеспечения.

УДК 004.414.38
ББК 16.332

ISBN 2410-4655

СОДЕРЖАНИЕ

Условные обозначения:

 важно

 обратить внимание (заметка)

 определение

 задание

 пример

 теоретические сведения

Введение

Цель учебной дисциплины «Технологии проектирования сложных информационных систем» – подготовка специалиста, владеющего теоретическими знаниями в области анализа и проектирования информационных систем и практическими навыками разработки программных решений, их эффективной реализации на объектно-ориентированных языках программирования.

Практическая часть по дисциплине «Технологии проектирования сложных информационных систем» включает 12 лабораторных работ, направленных на приобретение практических навыков по управлению разработкой программного обеспечения в рамках индивидуальной проектной деятельности на этапах **планирования, анализа и моделирования, проектирования, конструирования и внедрения**.

Лабораторная работа № 1	Планирование и организация проекта по разработке программного средства. Организация инфраструктуры разработки проекта
Лабораторная работа № 2	Разработка экономического обоснования проекта. Построение моделей AS-IS и TO-BE
Лабораторная работа № 3	Определение функционального назначения и контекста, составление обзора продукта. Формирование технического задания на разработку программного средства
Лабораторная работа № 4	Анализ требований и разработка спецификации требований к программному средству
Лабораторная работа № 5	Проектирование пользовательского интерфейса. Выбор технологий для реализации программного средства
Лабораторная работа № 6	Организация хранения данных. Выбор и описание архитектуры программного средства
Лабораторная работа № 7	Описание и разработка алгоритмов, реализующих бизнес-логику программного средства
Лабораторная работа № 8	Описание динамических аспектов поведения объектов системы
Лабораторная работа № 9	Программная реализация клиентской части программного средства
Лабораторная работа № 10	Программная реализация серверной части программного средства
Лабораторная работа № 11	Разработка тест-кейсов и юнит-тестов для проверки работоспособности программного средства
Лабораторная работа № 12	Внедрение программного средства. Оценка качества разработанного программного средства

Практическая часть представлена двумя разделами.

Первый раздел включает в себя краткие теоретические сведения и методические указания по выполнению **лабораторных работ № 1 – 4** и представлен в методическом пособии «**Разработка программного обеспечения: планирование, анализ и моделирование**»[1] по дисциплине «Технологии проектирования сложных информационных систем».

[1] Тонкович, И.Н., **Разработка программного обеспечения: планирование, анализ и моделирование**: метод. пособие по дисциплине «Технологии проектирования сложных информационных систем» для студентов очной формы обучения учреждений высшего образования направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте) / И. Н. Тонкович, А. В. Шелест. – Репозиторий БГУИР, 2023. – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/50325.

Представленное учебное пособие включает второй раздел, в котором рассматриваются краткие теоретические сведения и методические указания по выполнению **лабораторных работ № 5 – 12** по дисциплине «Технологии проектирования сложных информационных систем», направленных на приобретение практических навыков по управлению разработкой программного обеспечения в рамках проектной деятельности **на этапах проектирования, конструирования и внедрения**.

Все выполняемые лабораторные работы связаны между собой содержательно в рамках индивидуальной темы проекта и последовательностью выполнения.

В каждой лабораторной работе приводятся название работы, цель, перечень практических заданий, рекомендации по их выполнению, краткие теоретические выкладки, требования к содержанию отчета, контрольные вопросы. Обращено внимание на основные сложности, связанные с ее выполнением, а также на возможные типичные ошибки. Характерна взаимосвязь лабораторных работ – каждая последующая использует материал и результаты выполнения предыдущей.

Методически лабораторные работы практикума построены на принципах проектного метода обучения, основное назначение которого состоит в предоставлении студентам возможности самостоятельно конструировать свои знания в процессе реализации проектного решения по разработке программного средства, что требует интеграции знаний из различных предметных областей.

Преимуществом такой деятельности является возможность оптимально сочетать исследовательский и практико-ориентированный характер учебной деятельности, что представляется достаточно важным при подготовке специалиста программиста-бизнес-аналитика.

Представленное учебное пособие предназначено для студентов очной формы обучения направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте). Может быть использовано студентами заочной и дистанционной форм обучения и будет полезно всем, чья профессиональная деятельность связана с разработкой программного обеспечения.

Постановка задачи

Общая постановка задачи на выполнение лабораторных работ – реализовать проектное решение по разработке программного средства (ПС) в соответствии с требованиями, представленными на схеме (рисунок 1).

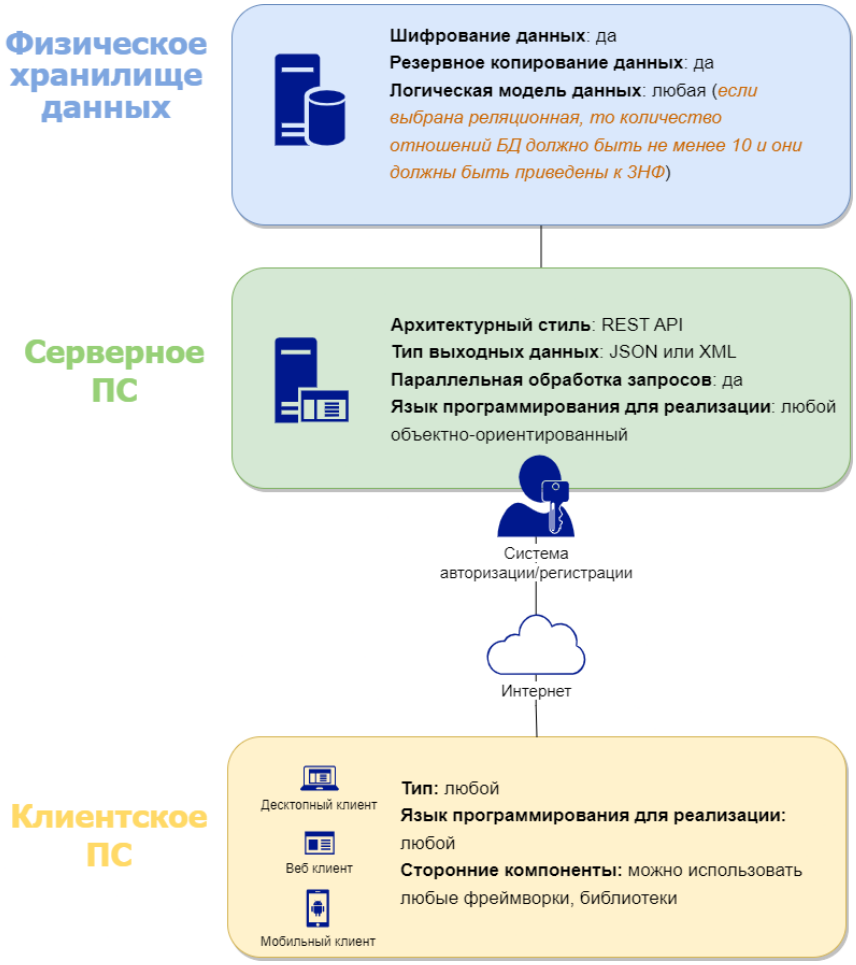


Рисунок 1 – Требования к программному средству

Вариант задания для выполнения лабораторных работ № 5 – 12 определяется вариантом задания для выполнения лабораторных работ № 1 – 4 (см. Приложение Б методического пособия «Разработка программного обеспечения: планирование, анализ и моделирование») и изменению не подлежит.

Требования к защите лабораторных работ

Подготовка, оформление и защита лабораторных работ регламентированы Порядком подготовки, оформления и защиты лабораторных работ студентами I ступени высшего образования от 26.06.2013.

Организация и проведение лабораторных занятий

Подготовка к лабораторным занятиям осуществляется студентами самостоятельно и заблаговременно. В процессе подготовки студент должен

усвоить теоретический материал, относящийся к лабораторной работе, изучить и ясно представлять себе содержание и порядок выполнения лабораторной работы.

Выполнение лабораторных работ производится **в течение занятия** в составе подгруппы. После выполнения лабораторной работы студенты предъявляют преподавателю результаты, **оформленные в виде отчета**.

В случае пропуска студентом лабораторного занятия без уважительной причины, он должен отработать пропущенные занятия в соответствии с Приказом «Об организации проведения повторной текущей и итоговой аттестации…» №284 от 24.11.2017 пункт 1 приложение 2. Студентам, пропустившим лабораторные занятия по уважительной причине[2], **в порядке исключения** преподавателем или деканатом может быть разрешено выполнение лабораторных работ в течение семестра, например, с другой группой.

Студенты, не защитившие лабораторные работы по учебной дисциплине, **к текущей аттестации по данной дисциплине не допускаются**, как не выполнившие график образовательного процесса.

Оформление отчета и защита лабораторных работ

Содержание и структура отчета определяются требованиями и методическими рекомендациями к лабораторной работе.

Отчет по выполненной лабораторной работе должен соответствовать следующим требованиям:

а) оформление отчета осуществляется согласно требованиям СТП БГУИР 01-2017 «Дипломные проекты (работы)»;

б) название файла отчета формируется согласно шаблону: **Лабораторная работа №X**, где X – порядковый номер лабораторной работы;

в) титульный лист отчета по лабораторной работе оформляется согласно **приложению А**;

г) **в обязательном порядке** на титульном листе отчета проставляются подпись студента, выполнившего лабораторную работу, и дата ее защиты;

д) **оформленный отчет** по лабораторной работе следует разместить в папке с названием семестра обучения на Google-диске[3], соблюдая указанную структуру директорий (рисунок 2).

В случае невыполнения одного из вышеуказанных требований студент будет не допущен к защите лабораторных работ.

Защита лабораторных работ проводится во время **очных занятий** согласно графику защиты лабораторных работ, а также **в сроки, установленные кафедрой**.

[2] Уважительным считается пропуск, подтверждаемый документально

[3] Ссылку на доступ к папке уточнить у преподавателя

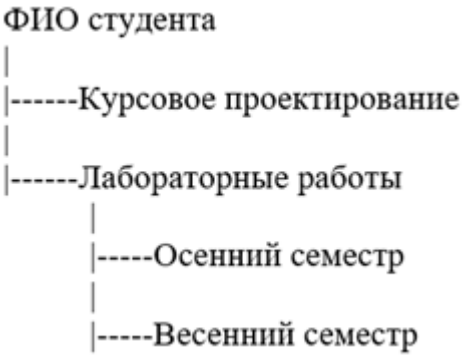


Рисунок 2 – Организация директорий в папке на Google-диске

Оценка результатов защиты лабораторных работ

По результатам защиты лабораторной работы студенту выставляется отметка по 10-ти балльной шкале. В случае получения студентом отметки 4 балла и более работа считается защищенной.

Оценка результатов выполнения лабораторной работы осуществляется в соответствии с Положением «О порядке организации и проведении текущей аттестации студентов…» №03-2013/03-0015 от 11.12.2013 приложение 1.

Проведение повторной защиты лабораторных работ

В ситуации получения студентом отметки **ниже 4 баллов** по результатам защиты лабораторной работы ему необходимо подать заявление на имя декана факультета с просьбой разрешить повторно отчитаться по результатам выполнения лабораторной работы. Декан факультета рассматривает поданное заявление, на его основе определяет возможность и стоимость повторной аттестации в соответствии с действующими в университете ценами на платные услуги и сроки ее проведения.

После предъявления квитанции об оплате в деканат, студенту выдается разрешение, которое дает право отработать пропущенные лабораторные работы или повторно отчитаться по их результатам.

Преподаватель на основании разрешения деканата назначает время и место выполнения студентом учебной работы, организует ее и осуществляет аттестацию студента. При отработке лабораторных работ преподаватель, ведущий лабораторный практикум, формирует группы из задолжников (5-7 человек) и сообщает им дату, аудиторию и время отработки лабораторных работ или повторной защиты. Результаты аттестации преподаватель в установленном порядке представляет в деканат.

Дополнительно

Принятие решений в случаях, не предусмотренных Порядком подготовки, оформления и защиты лабораторных работ осуществляют в пределах своей компетенции Учебно-методическое управление университета, дека-

наты, заведующие кафедрами и преподаватели, ведущие лабораторные занятия, на основании Положения об учреждении высшего образования, Устава БГУИР и других нормативных актов.

Организация хранения исходного кода

Перед выполнением лабораторных работ необходимо создать публичный репозиторий, используя любую систему контроля версий (СКВ). Стоит учесть, что выбранная СКВ должна поддерживать возможность обеспечения доступа к хранилищу исходного кода незарегистрированным в ней пользователям (это необходимо для предоставления возможности проверки исходного кода преподавателю).

По мере выполнения лабораторных работ необходимо размещать исходный код программы в выбранной СКВ (**ссылку на аккаунт с исходным кодом необходимо приводить в начале отчета по каждой лабораторной работе**). Название репозитория СКВ должно соответствовать маске [4] **: FamiliIO_№группы.**

В репозитории нужно создать две ветки: master и develop. В ветке master будет находиться полностью рабочий код разрабатываемого программного средства, а ветке develop – код, находящийся в стадии разработки.

[4] Все репозитории, название которых не соответствует маске, не будут учитываться при защите лабораторных работ

Лабораторная работа № 5

Проектирование пользовательского интерфейса.
Выбор технологий для реализации программного средства

Цели выполнения лабораторной работы:

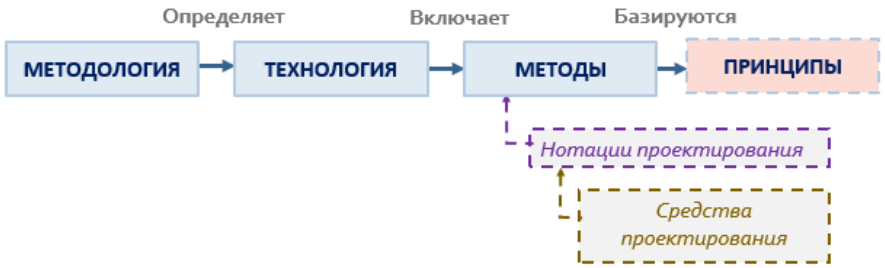
- построить карту макетов пользовательского интерфейса
- выбрать стек технологий для реализации проекта

Задание	Этапы выполнения задания
1 Разработать пользовательский интерфейс программного средства	1 Построить карту макетов пользовательского интерфейса программного средства 2 Разработать макеты всех элементов пользовательского интерфейса
2 Выбрать стек технологий для разработки проекта	1 Проанализировать технологии, используемые для реализации схожих задач 2 Выбрать наиболее подходящие технологии для разработки программного средства
3 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 MindMap логики действий пользователя при работе с программным средством 3 Планы расположения элементов пользовательского интерфейса 4 Дизайн элементов пользовательского интерфейса 5 Анализ готовых решений 6 Описание стека технологий для разработки проекта с обоснованием их выбора 7 Выводы

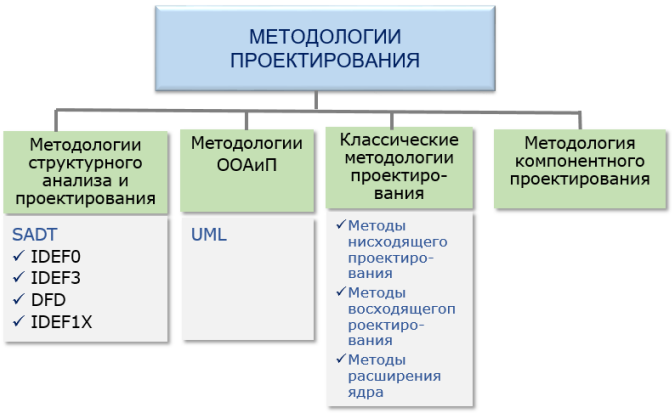
Краткие теоретические сведения и методические указания к Заданию 1

Общие сведения об этапе проектирования программного обеспечения

ISO/IEC/ IEEE 24765	**Проектирование** – часть стадии жизненного цикла системы, процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части.
SWEBoK	**Проектирование** – программный дизайн **(Software Design)** (как результат деятельности по проектированию) должен описывать архитектуру ПО, то есть представлять декомпозицию программной системы в виде организованной структуры компонентов и интерфейсов между компонентами.

Проектирование	❑ Процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного результата. ❑ Инженерная деятельность в рамках жизненного цикла ПО, в которой надлежащим образом анализируются требования для создания описания внутренней структуры ПО, являющейся основой для конструирования программного обеспечения.
Ступени (шаги) проектирования	❑ Предварительное (архитектурное) проектирование – формирует абстракции архитектурного уровня (архитектура программ и данных). Включает три типа деятельности: структурирование системы; моделирование управления; декомпозиция подсистем на модули. ❑ Детальное проектирование – уточняет абстракции, добавляет подробности алгоритмического уровня (структуры и алгоритмы программ). ❑ Интерфейсное проектирование.
Области проектирования	❑ Проектирование объектов данных, которые будут реализованы в базе данных. ❑ Проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным. ❑ Учет конкретной среды или технологии, а именно: топологии сети, конфигурации аппаратных средств, используемой архитектуры, параллельной обработки, распределенной обработки данных и т.п.
Составляющие проектирования	
МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ	Предлагает общие принципы проектирования, определяет подходы (концептуальную модель) к оценке и выбору варианта системы, последовательность стадий и этапов проектирования и в конечном итоге позволяет выбрать метод проектирования.
Цель методологии проектирования	Регламентировать процесс проектирования и: ❑ обеспечить создание системы, отвечающей целям и задачам организации, а также предъявляемым требованиям по автоматизации деловых процессов заказчика; ❑ гарантировать создание системы с заданным качеством в заданные сроки и в рамках установленного бюджета проекта;

- поддерживать удобную дисциплину сопровождения, модификации и наращивания системы;
- обеспечивать преемственность разработки, т.е. использование в разрабатываемой системе существующей информационной инфраструктуры организации (задела в области информационных технологий).



ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ

Совокупность технологических операций проектирования в их последовательности и взаимосвязи, приводящая к разработке проекта системы.

Требования к технологии проектирования

- Поддерживать полный ЖЦ системы и обеспечивать гарантированное достижение целей ее разработки с заданным качеством и в установленное время.
- Обеспечивать возможность декомпозиции системы на составные части.
- Обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами.
- Обеспечивать минимальное время получения работоспособной системы.
- Предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта.
- Обеспечивать независимость выполняемых проектных решений от средств реализации системы – СУБД, ОС, языка и системы программирования.

Примеры технологии проектирования

- CASE-технология. Представляет собой совокупность методов проектирования системы, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех стадиях разработки, сопровождения системы и разрабатывать приложения в соответствии с информационными потребностями пользователей.
- Технология проектирования RUP.
- Технология проектирования DATARUN.

Классы технологии программирования

МЕТОДЫ ПРОЕКТИРОВАНИЯ

Представляют собой совокупность:

- концепций и теоретических основ;
- нотаций, используемых для построения моделей статической структуры и динамики поведения проектируемой системы;
- процедур, определяющих практическое применение метода;

Метод конкретизирует порядок разработки отдельных элементов, комплексов задач, подсистем и системы в целом.

Выделяют:

- Функционально-ориентированные (структурные) методы. Базируются на структурном анализе, структурных картах, Dataflow-диаграммах и др. Они ориентированы на идентификацию функций и их уточнение сверху-вниз, после чего проводится разработка диаграмм потоков данных и описание процессов.
- Объектно-ориентированные методы. Базируются на ключевых принципах ООП: наследование, полиморфизм и инкапсуляция, а также абстрактные структуры данных и отображение объектов.
- Ориентированные на структуры данных. Базируются на методе Джексона (Jackson) и используются для задания входных и выходных данных структурными диаграммами.
- Компонентное проектирование. Ориентировано на использование и интеграцию компонентов (особенно компонентов повторного использования) и на их интерфейс, обеспечивающий взаимодействие компонентов; является базисом других видов программирования, в том числе сервисно-ориентированного, в котором группы компонентов обеспечивают функциональный сервис.
- Другие методы. Относятся: формальные, точные и трансформационные методы, а также UML для моделирования архитектурных решений с помощью диаграмм.

НОТАЦИИ ПРОЕКТИРОВАНИЯ	Позволяют представить артефакты ПО и его структуру, а также поведение системы. Выделяют два типа нотаций: структурные и поведенческие (однако существует множество различных их представлений): ❑ **Структурные нотации** используются для представления структурных аспектов проектирования, компонентов и их взаимосвязей, элементов архитектуры и их интерфейсов. К ним относятся формальные языки спецификаций и проектирования: ADL, UML, ERD, IDL, классы и объекты, компоненты и классы, Use Case Driven и др. Нотации включают языки описания архитектуры и интерфейса, диаграммы классов и объектов, диаграммы сущность-связь, компонентов, развертывания, а также структурные диаграммы и схемы. ❑ **Поведенческие нотации** отражают динамический аспект поведения систем и их компонентов. Таким нотациям соответствуют диаграммы: Data Flow, Decision Tables, Activity, Colloboration, Pre-Post Conditions, Sequence, таблицы принятия решений, формальные языки спецификации, языки проектирования PDL и др.
СРЕДСТВА ПРОЕКТИРОВАНИЯ	Инструментальные средства проектирования, поддерживающие метод проектирования.
CASE-средство	Программное средство, поддерживающее процессы ЖЦ ПО (определённые в стандарте ISO/IEC 12207:1995), включая анализ требований к системе, проектирование прикладного ПО и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы.

Важной частью этапа проектирования является **проектирование пользовательского интерфейса**.

Проектирование пользовательского интерфейса – это создание своеобразной тестовой версии приложения. Это начальный этап разработки, когда выполняется распределение функций приложения по экранам, определяется графическое оформление будущего программного средства, содержимое, элементы управления и их поведение. Как результат, получается динамичный прототип интерфейса, который можно использовать для тестирования юзабилити или начала разработки приложения.

В рамках лабораторной работы проектирование пользовательского интерфейса программного средства будет осуществляться с использованием концепций UX (англ. user experience – пользовательский опыт) и UI (англ. user interface – пользовательский интерфейс) по следующему алгоритму:

1 Составить MindMap (англ. mind map – карта разума, рисунок 3), чтобы описать логику действий пользователя в приложении. Также можно рассмотреть и создание CJM[5] (англ. customer journey map – карту пользовательских путей).

2 Разработать планы расположения элементов на экранах (рисунок 4).

3 Подобрать контент, который будет представлен в ПС (картинки, видео, текст).

4 Разработать дизайн элементов пользовательского интерфейса (выбор типографики и цветовой гаммы, создание логотипа, подбор аудио и графических эффектов). Примеры дизайнов представлены на рисунках 5 и 6.

5 Разместить разработанный пользовательский интерфейс на тестовом домене (устройстве) и провести тестирование (проверить верстку, корректное отображение элементов и содержимого и прочее).

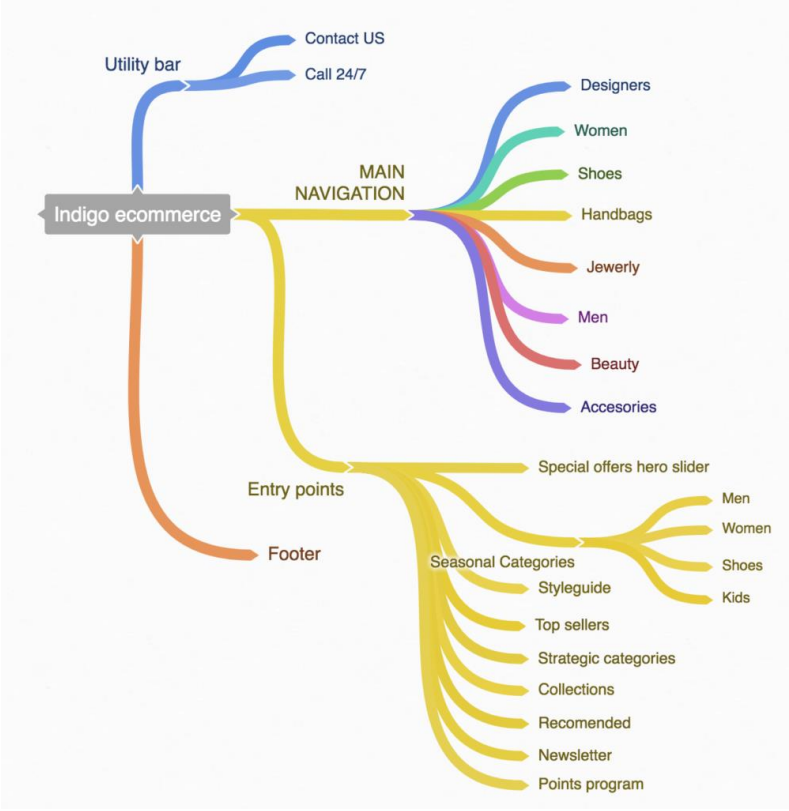


Рисунок 3 – Пример MindMap для визуализации пользовательского опыта[6]

[5] Пример представлен по ссылке: https://www.unisender.com/ru/blog/wp-content/uploads/2020/09/2-cjm.jpg

[6] Изображение представлено по ссылке: https://www.infolob.com/wp-content/uploads/2020/02/Mind-map-997x1024.png

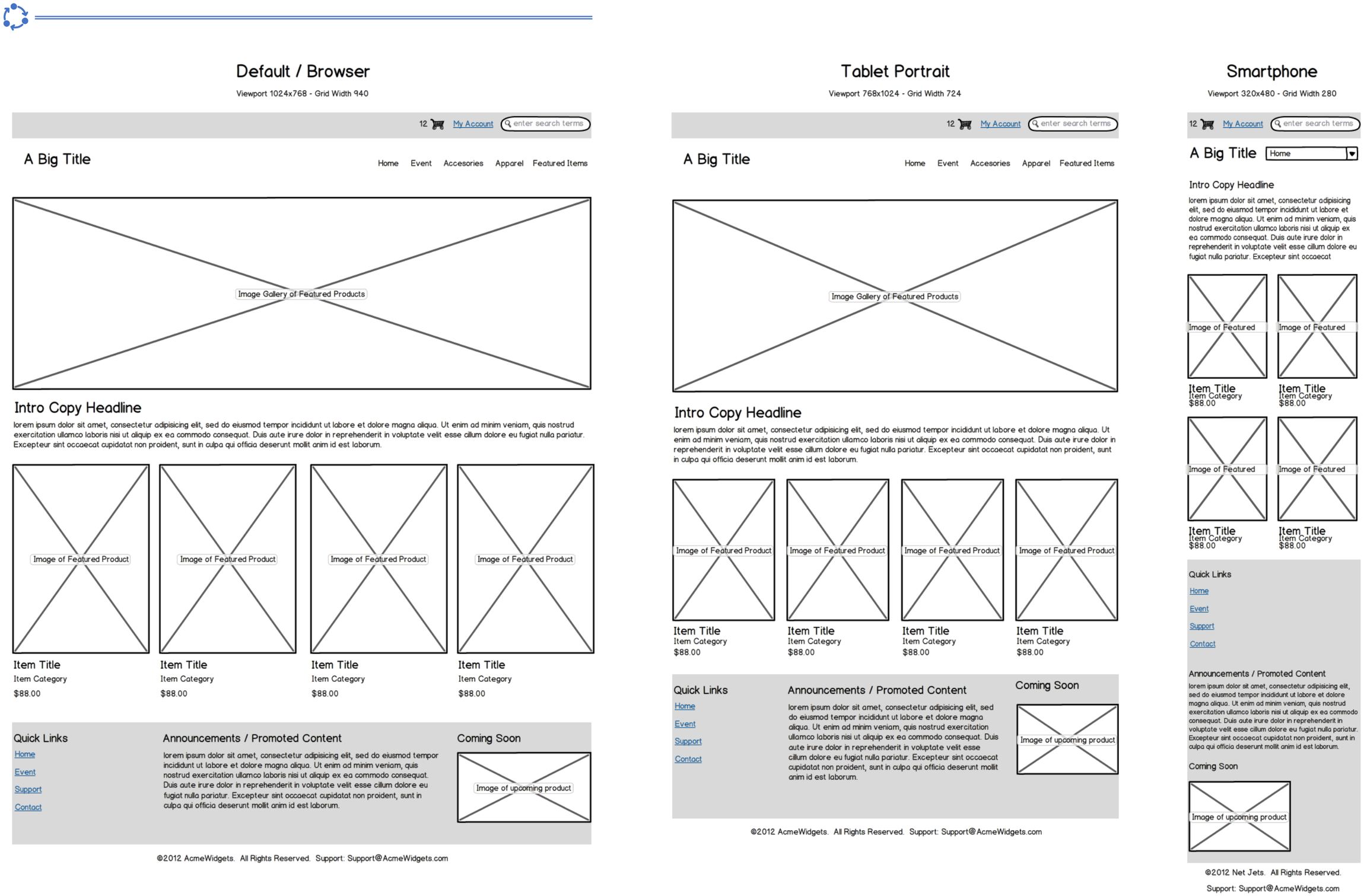


Рисунок 4 – Пример плана расположения элементов для различных типов устройств[7]

[7] Изображение представлено по ссылке: https://file.mockplus.com/image/2017/11/e8bbb60f-f2da-49cf-9c3b-64dbefe77b27.png

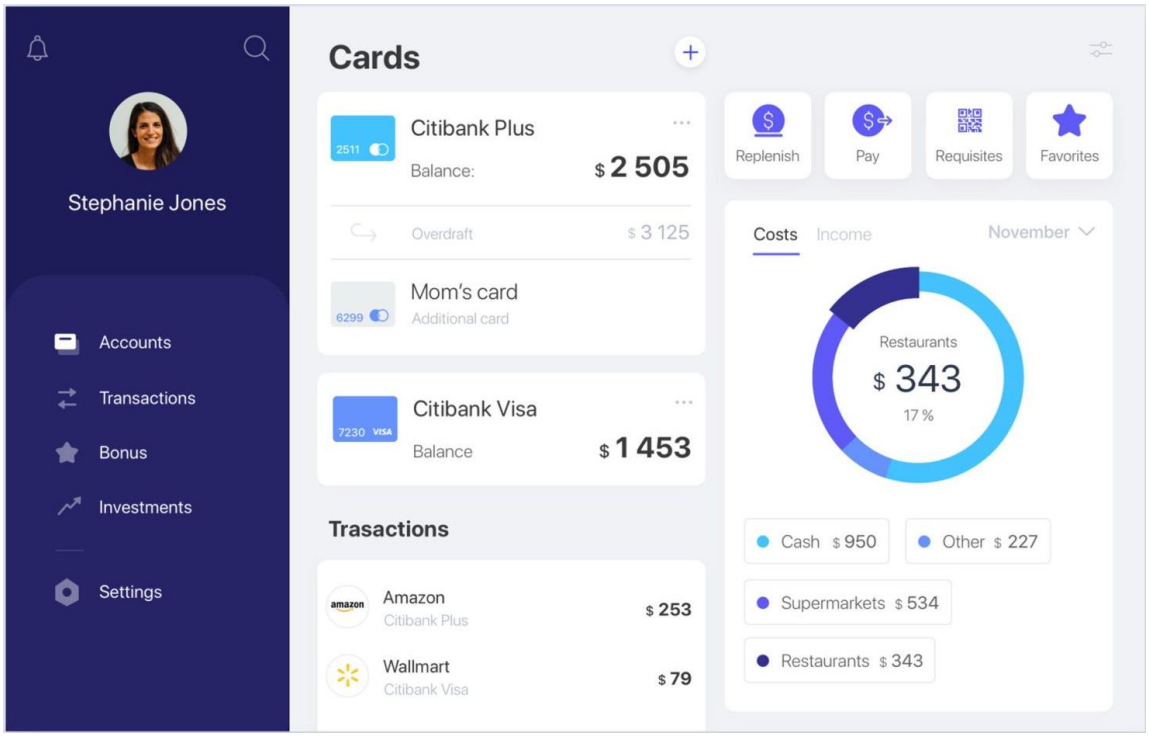


Рисунок 5 – Пример пользовательского интерфейса веб-приложения[8]

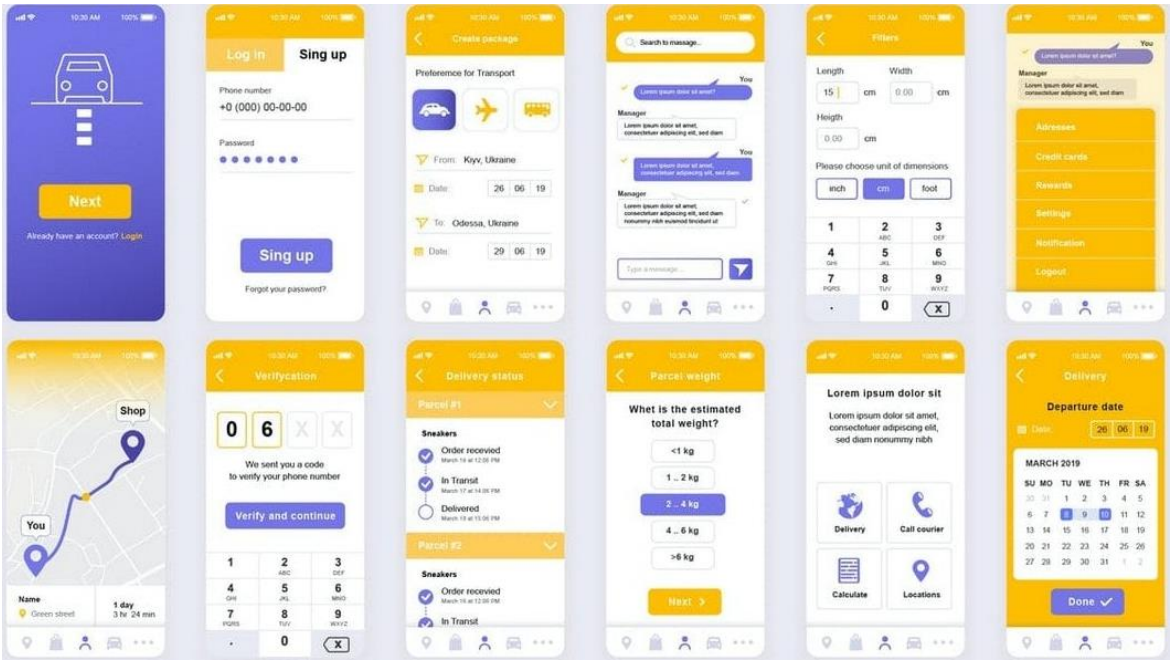


Рисунок 6 – Пример пользовательского интерфейса мобильного приложения[9]

[8] Изображение представлено по ссылке: https://miro.medium.com/max/3200/1*YKbwwEA84V3st8fIlfU0Hw.jpeg

[9] Изображение представлено по ссылке: https://designshack.net/wp-content/uploads/Delivery-Mobile-App-UI-Templates-Kit.jpg

Принципы и примеры построения планов расположения элементов на страницах достаточно подробно рассмотрены в источниках [1-3].

Для проектирования пользовательского интерфейса можно использовать, например, следующее программное обеспечение:

- Figma (https://www.figma.com/);
- Whimsical (https://whimsical.com/);
- inVision (https://www.invisionapp.com/home);
- Adobe XD (https://www.adobe.com/ru/products/xd.html);
- Zeplin (https://zeplin.io/);
- Draw.io (https://www.drawio.com/);
- MindMeister (https://www.mindmeister.com/);
- MindMup (https://www.mindmup.com/);
- Miro (https://miro.com/mind-map/)

Краткие теоретические сведения и методические указания к Заданию 2

После разработки графического пользовательского интерфейса следует провести **анализ необходимости использования готовых решений** (пример анализа приведен ниже), которые уже используются для решения поставленных или схожих с поставленными задачами бизнеса. Использование готовых решений позволяет получить более качественный код в краткие сроки с меньшими затратами, что влечет за собой увеличение прибыли компании и повышения уровня удовлетворенности заказчиков. Помимо этого, выгоду от использования готовых решений получают и работники, т.к. они могут уделить больше времени на реализацию нетривиальных задач заказчика.

После того, как проанализированы готовые решения, осуществляется выбор компонентов и технологий для реализации ПС.

Результаты анализа необходимости использования готовых решений следует представить в виде таблицы с представленными критериями, с помощью которых будет проводиться сравнение и на основании которых в последующем будет производиться выбор технологий для реализации ПС.

Пример анализа готовых решений

Задача прогнозирования величины урожая сводится к предсказанию значений временного ряда. На основании проведенного анализа литературных источников и Интернет-ресурсов был сделан вывод, что на текущий момент для прогнозирования значений временного ряда применяются математические модели под названием искусственные нейронные сети (ИНС). Для построения структуры и моделирования работы таких моделей можно использовать большинство высокоуровневых языков программирования. Однако,

среди множества языков разработчики выделяют язык Python, так как именно для него доступны основные версии API различных фреймворков для работы с ИНС. На основании изучения в сети Интернет-ресурсов, посвященных анализу данных и построению ИНС, было принято решение, в качестве потенциальных фреймворков для осуществления прогнозирования рассмотреть следующие, наиболее распространенные фреймворки: TensorFlow, Keras, PyTorch.

Для того, чтобы провести сравнительный анализ и выявить фреймворк, который лучше всего подходит для решения поставленных задач, были выбраны следующие характеристики: скорость работы, тестовые модели для обучения, отладка, набор данных, популярность. Параметр «популярность» был выбран исходя из соображения, что чем более популярен фреймворк, тем большее количество учебных материалов о нем есть в свободном доступе. Результаты сравнения фреймворка представлены в таблице 1.

Таблица 1 – Сравнительный анализ фреймворков для работы с ИНС

Характеристика	**Фреймворк**		
	TensorFlow	**Keras**	**PyTorch**
скорость работы	высокая	низкая	высокая
тестовые модели для обучения	да	да	да
отладка	обладает наиболее широким набором функций для отладки среди выбранных фреймворком	требуется редко ввиду простоты модуля	присутствует множество функций для отладки кода
набор данных	предназначен для работы с большими объема данных без потери в производительности	оптимален для небольших наборов данных	предназначен для работы с большими объема данных без потери в производительности
популярность	занимает второе место по популярности среди выбранных фреймворков	наиболее популярен среди выбранных фреймворков	занимает третье место по популярности среди выбранных фреймворков

На основании анализа предметной области и данных таблицы 1 можно сделать вывод, что оптимальным фреймворком для решения задачи прогнозирования величины урожая является TensorFlow.

Далее осуществляется **выбор компонентов и технологий для реализации** ПС. Данный этап является одним из важных этапов в разработке, т.к. позволяет сократить время и сроки разработки.

При обосновании выбора технологий и компонентов для реализации, необходимо учитывать особенности функциональности ПС (например, необходимо, чтобы некоторые функции выполнялись параллельно), а также инфраструктуры (например, версия ОС или Android), в которой разработанное ПС будет осуществлять работу.

Пример обоснования выбора компонента

Исходя из требований, определенных в техническом задании, а также результатов анализа готовых решений, было принято решение, что в качестве основного языка разработки будет использоваться объектно-ориентированный язык Java, так как он позволяет разрабатывать приложения независимо от конечной пользовательской архитектуры. Это обусловлено тем, что для выполнения кода приложения на Java требуется лишь JVM (Java Virtual Machine – виртуальная машина Java). Байт-код, который получается в результате компиляции приложений написанных на Java, может быть запущен везде именно благодаря виртуальной машине…

Также стоит описать конкретные «особенные» элементы компонента или технологии, которые отсутствуют в других похожих компонентах и которые будут использоваться в процессе разработки. Например, «… для реализации полиморфизма в Java не требуется указание дополнительных модификаторов, поскольку Java поддерживает полиморфизм по умолчанию…».

Контрольные вопросы к лабораторной работе № 5

1 Какие задачи решают на этапе проектирования ПО?

2 Что такое пользовательский интерфейс?

3 Что представляет собой процесс проектирования пользовательского интерфейса?

4 Какие компоненты включает пользовательский интерфейс?

5 Каковы критерии качественного интерфейса пользователя?

6 Что такое графический интерфейс пользователя и какие элементы включает?

7 Какие бывают виды графического интерфейса?

8 Какие существуют популярные инструменты для создания прототипов и дизайна интерфейса?

Лабораторная работа № 6

Организация хранения данных.
Выбор и описание архитектуры программного средства

Цели выполнения лабораторной работы:

- выбрать модель данных и организовать хранение данных
- спроектировать архитектуру программного средства
- описать поведение сложных объектов программной системы

Задание	Этапы выполнения задания
1 Организовать хранение данных	1 Разработать и описать структуру данных для хранения информации
2 Спроектировать и описать архитектуру разрабатываемого программного средства	1 Посредством диаграммы классов показать используемые в архитектуре программного средства паттерны проектирования 2 При помощи диаграмм компонентов и размещения показать дизайн системы и расположение компонентов по узлам системы (если программное средство обладает множеством классов, узлов или других компонентов, тогда дополнительно строится диаграмма пакетов)
3 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Описание структуры данных (независимо от модели данных должны быть представлены схема организации структуры для хранения данных и описаны использованные типы данных) 3 Концептуальная модель данных (только для реляционной модели данных) 4 Логическая модель данных и обоснование соответствия ее правилам 3НФ (только для реляционной модели данных) 5 Физическая модель данных (только для реляционной модели данных) 6 Описание используемых паттернов проектирования с использованием UML диаграмм классов, размещения и/или пакетов 7 Моделирование поведения объектов программной системы 8 Моделирование взаимодействия объектов программной системы 9 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

Выбор модели данных для хранения информации зависит от поставленной задачи, требований заказчика и множества других факторов. На сегодняшний день существует большое количество моделей данных, наиболее популярной среди которых является реляционная. В контексте данных методических указаний будет рассмотрено моделирование хранения данных с использованием реляционной модели данных.

В отчете по лабораторной работе (независимо от того, какая модель данных была выбрана) в обязательном порядке представить:

- схему организации данных;
- описание выбранных типов данных в табличном виде;
- описание всех дополнительных элементов. Под дополнительными элементами подразумеваются любые инструкции языков определения и манипулирования данными для выбранной СУБД.

Информационная модель данных предназначена для представления семантики предметной области в терминах субъективных средств описания – сущностей, атрибутов, идентификаторов сущностей, супертипов, подтипов и т.д.

Информационная модель предметной области базы данных содержит следующие основные конструкции:

– диаграммы «сущность-связь» (Entity-Relationship Diagrams);
– определения сущностей;
– уникальные идентификаторы сущностей;
– определения атрибутов сущностей;
– отношения между сущностями;
– супертипы и подтипы.

Для разработки информационной модели предметной области, основываясь на архитектуре ANSI/SPARC, выделяют следующие уровни абстракции базы данных (БД): концептуальный, логический и физический уровни (рисунок 7).

Разработка информационной модели включает следующие этапы проектирования БД:

1 *Концептуальное (инфологическое) проектирование (Conceptual design)*. На данном этапе осуществляется анализ предметной области и выявление основных сущностей предметной области, а также связей между ними. Итогом данного этапа проектирования является концептуальная модель БД в виде ER-модели.

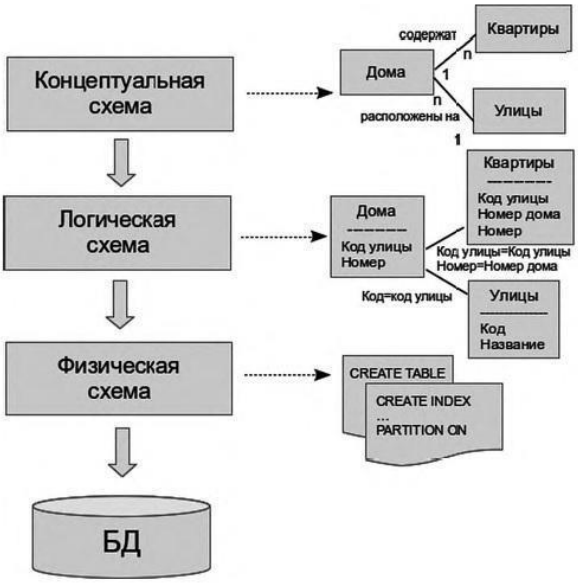


Рисунок 7 – Уровни абстракции БД

Обычно концептуальная модель включает в себя:

– описание информационных объектов или понятий предметной области и связей между ними;

– описание ограничений целостности, то есть требований к допустимым значениям данных и к связям между ними.

2 *Логическое (даталогическое) проектирование (Logical Design)*. В ходе данного этапа происходит преобразование концептуальной модели БД в логическую. Отличительной особенностью логической схемы БД является указание атрибутов сущностей и их типа (ключевые, неключевые, многозначные, производные, простые, составные). Результатом данного этапа проектирования является логическая схема БД, выполненная в соответствии с выбранной нотацией.

Отдельно стоит отметить процесс нормализации данных, который осуществляется перед этапом физического проектирования. Как отмечает К. Дж. Дейт, общее назначение процесса нормализации заключается в [4, с. 509]:

– исключении некоторых типов избыточности;

– устранении некоторых аномалий обновления;

– разработке проекта базы данных, который является достаточно «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;

– упрощении процедуры применения необходимых ограничений целостности.

На каждом этапе нормализации в БД добавляется новая таблица.

3 *Физическое проектирование (Physical Design).* Это последний этап проектирования БД. В рамках данного этапа осуществляется преобразование логической схемы в схему БД для конкретной СУБД. Различные СУБД могут

включать в себя ограничения на именование объектов БД, ограничения на поддерживаемые типы данных и прочее. Кроме того, при физическом проектировании специфика СУБД включает выбор решений, связанных с физической средой хранения данных (выбор методов управления дисковой памятью, разделение БД по файлам и устройствам, методов доступа к данным), создание индексов и так далее. Следствием физического проектирования может быть ER-диаграмма БД с указанием конкретных типов данных или скрипт генерации БД.

Процесс нормализации, осуществляющийся перед этапом физического проектирования, является отдельной ступенью в проектировании БД.

Нормализация – это процесс структурирования реляционной базы данных в соответствии с правилами нормальных форм для уменьшения избыточности данных и улучшения целостности данных.

Для лучшего понимания сущности процесса нормализации данных приведем основные определения.

Определение 1. Атомарность – это степень детализации и структурирования информации в базе.

Определение 2. Функциональная зависимость. В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X -> R.Y.

Определение 3. Полная функциональная зависимость. Функциональная зависимость R.X -> R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение 4. Транзитивная функциональная зависимость. Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z -> R.X.

Определение 5. Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа.

Определение 6. Два или более атрибута **взаимно независимы**, если ни один из этих атрибутов не является функционально зависимым от других.

Определение 7. Отношение находится в **первой нормальной форме (1NF)** при атомарности значений всех его атрибутов.

Определение 8. Отношение находится во **второй нормальной форме (2NF)**, если оно находится в *первой нормальной форме* и каждый неключевой атрибут находится в полной функциональной зависимости от ключа.

Определение 9. Отношение находится в **третьей нормальной форме (3NF)**, если данное отношение находится во *второй нормальной форме* и каждый неключевой атрибут находится в нетранзитивной зависимости от первичного ключа.

Более подробное описание проблем и процесса нормализации данных представлено в презентации по проектированию БД[10].

Пример проектирования информационной модели (аренда помещений)

Арендодатель хранит информацию о помещениях и арендаторах (таблицы 2-3).

Таблица 2 – Данные о помещениях

Инвентарный номер помещения	Этаж	Номер помещения[11]	Площадь, м2	Количество окон	Наличие кондиционера (да/нет)	Мебель	Стоимость аренды в месяц, $
1001256	7	284	85	4	да	компьютерные столы – 6 шт; письменные столы – 2 шт; кресла с поворотным механизмом и регулятором высоты – 10 шт; шкаф-купе –1 шт	300
1001278	17	245	175	8	да	компьютерные столы – 6 шт; письменные столы – 2 шт; кресла с поворотным механизмом и регулятором высоты – 10 шт; шкаф-купе – 1 шт; диван – 3 шт; журнальный столик – 1 шт; подставки для техники – 3 шт; кухонный гарнитур – 1 шт; микроволновка – 1 шт; чайник электрический – 1 шт; кофемашина – 1 шт	800
1001256	1	140	69	4	да	компьютерные столы – 6 шт; письменные столы – 2 шт; кресла с поворотным механизмом и регулятором высоты – 10 шт; шкаф-купе – 1 шт	275
1001257	13	95	65	2	нет	компьютерные столы – 6 шт; письменные столы – 2 шт; кресла с поворотным механизмом и регулятором высоты – 10 шт; шкаф-купе – 1 шт	200

10 Ссылка на презентацию: https://practicum.yandex.ru/blog/chto-takoe-normalizaciya-dannyh/

11 Условимся, что номер помещения зависит от этажа

Инвентарный номер помещения	Этаж	Номер помещения[11]	Площадь, м2	Количество окон	Наличие кондиционера (да/нет)	Мебель	Стоимость аренды в месяц, $
1001243	9	426	133	7	да	компьютерные столы – 6 шт; письменные столы – 2 шт; кресла с поворотным механизмом и регулятором высоты – 10 шт; шкаф-купе – 1 шт	540

Таблица 3 – Данные об арендаторах и аренде помещений

Номер договора аренды	ФИО арендатора	Номер паспорта	Email	Номер для связи[12]	Инвентарный номер арендуемого помещения	Дата начала аренды	Срок аренды, дн
24123	Смирнов Александр Валерьевич	AB1870041	austin1988@gmail.com	+375291234567	1001256	02.05.2020	30
55342	Касперович Наталья Игоревна	KH2105396	torrance2006@hotmail.com	+375291876543	1001256	12.06.2020	30
78298	Шиманчук Дмитрий Александрович	MP4007108	margie.rolfs@gmail.com	+375299876543	1001278	02.09.2020	30

Разработка информационной модели предметной области начинается с этапа концептуального проектирования.

Этап концептуального проектирования. На первом этапе проектирования схемы БД необходимо выделить основные сущности предметной области, а также определить связи между ними. Из таблицы 2 можно выделить сущность ПОМЕЩЕНИЕ, а из таблицы 3 – АРЕНДАТОР и ДОГОВОР АРЕНДЫ. Представим данные сущности в виде ER-модели (рисунок 8).

12 Номер для связи зависит от номера паспорта

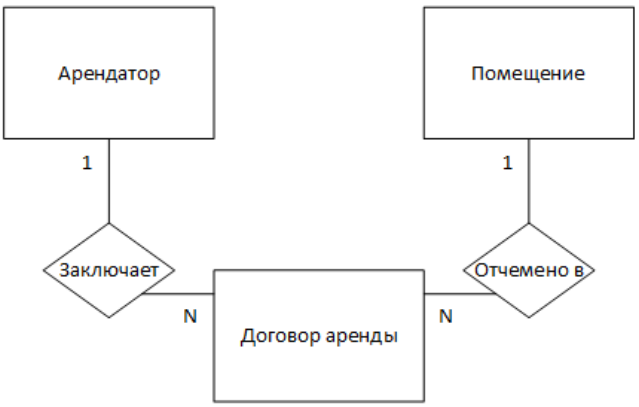


Рисунок 8 – ER-модель предметной области

Логическое проектирование. Следующим этапом проектирования является логическое проектирование. В рамках данного этапа нужно выделить атрибуты сущностей. Для представления модели БД на данном этапе воспользуемся нотацией Чена[13].

Рассмотрим сущность ПОМЕЩЕНИЕ. Из таблицы 2 видно, что ключевым атрибутом данной сущности является поле ИНВЕНТАРНЫЙ НОМЕР ПОМЕЩЕНИЯ. Остальные атрибуты являются простыми. Отображение сущности[14] ПОМЕЩЕНИЕ в нотации Чена представлено на рисунке 9.

Рисунок 9 – Сущность ПОМЕЩЕНИЕ

Теперь представим отображение сущностей АРЕНДАТОР и ДОГОВОР АРЕНДЫ. Так как таблица 3 одновременно содержит информацию двух сущностей, нужно отделить атрибуты одной сущности от другой. К сущности

[13] Дополнительная информация о модели сущность-связь представлена по ссылке https://foreva.susu.ru/courses/db/lecture3.pdf

[14] Для построения использовался сервис https://erdplus.com/, также можно воспользоваться MS Visio, выбрав в качестве шаблона «Нотация Чена для моделирования баз данных»

АРЕНДАТОР отнесем поля: Фамилия, Имя, Отчество, Номер паспорта, Номер для связи, Email. Оставшиеся поля отнесем к сущности ДОГОВОР АРЕНДЫ.

Стоит отметить тот факт, что однозначно идентифицировать различные кортежи сущности АРЕНДАТОР невозможно, так как в ней отсутствует ключевой атрибут. Логичным, на первый взгляд, могло бы быть выделение в качестве первичного ключа поля Номер паспорта, однако, данный атрибут на самом деле не является уникальным.

Другим вариантом может быть формирование составного первичного ключа из полей Фамилия, Имя, Отчество. Но даже эта комбинация полей может оказаться не единственной в своем роде. В таком случае необходимо использовать суррогатный первичный ключ, который получается путем добавления в сущность нового атрибута (в нашем случае поля ID_АРЕНДАТОРА). Итоговые представления сущностей АРЕНДАТОР и ДОГОВОР АРЕНДЫ показаны на рисунках 10 и 11 соответственно.

Рисунок 10 – Сущность АРЕНДАТОР

Итоговая логическая модель вместе со связями между указанными ранее сущностями показана на рисунке 12.

Прежде чем приступить к этапу физического проектирования оценим степень нормализации спроектированной логической схемы.

Основываясь на определении 7, проанализируем состав атрибутов. Как видно из схемы БД, все атрибуты являются атомарными, поэтому можно сделать вывод, что схема соответствует 1НФ.

Рисунок 11 – Сущность ДОГОВОР АРЕНДЫ

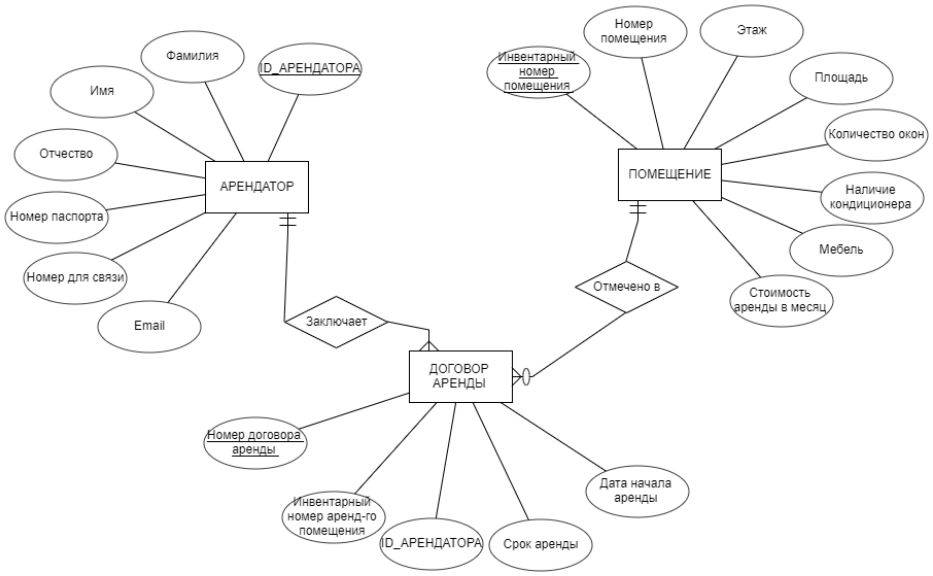


Рисунок 12 – Логическая схема БД

Для приведения схемы ко 2НФ необходимо, чтобы все неключевые атрибуты зависели только от всего составного первичного ключа, но не зависели ни от одной его части. Так как в сущностях отсутствуют составные первичные ключи, можно говорить о соответствии схемы БД 2НФ.

Теперь можно проверить, соответствует ли схема 3НФ. Для этого необходимо, чтобы все неключевые атрибуты нетранзитивно зависели от первичного ключа.

В сущности АРЕНДАТОР присутствует транзитивная зависимость между атрибутами ID_АРЕНДАТОРА и Номер для связи:

ID_АРЕНДАТОРА -> Номер паспорта Номер паспорта -> Номер для связи	⇒ ID_АРЕНДАТОРА -> Номер для связи

Такая же транзитивная зависимость существует между атрибутами Инвентарный номер помещения и Номер помещения сущности ПОМЕЩЕНИЕ:

Инвентарный номер помещения -> Этаж Этаж -> Номер помещения	⇒ Инвентарный номер помещения -> Номер помещения

Чтобы устранить данные зависимости, необходимо декомпозировать на две каждую из сущностей АРЕНДАТОР и ПОМЕЩЕНИЕ. В итоге декомпозиции получим новую схему БД (рисунок 13).

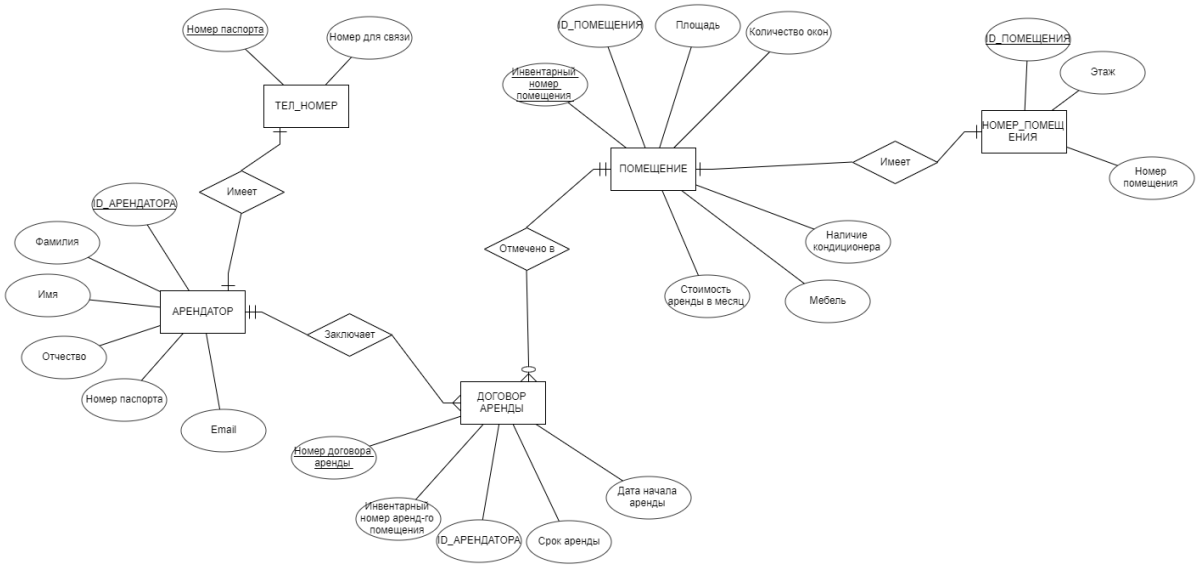


Рисунок 13 – Схема БД после декомпозиции

После удаления транзитивных зависимостей схема соответствует 3НФ.

Физическое проектирование. Теперь перейдем к последнему этапу – физическое проектирование. На данном этапе необходимо выбрать конкретную СУБД и преобразовать логическую схему БД в физическую. Для примера воспользуемся СУБД MySQL, а в качестве визуального средства проектирования будем использовать MySQL Workbench. В итоге преобразования получим физическую схему БД, представленную на рисунке 14.

Для демонстрации разработки информационной модели предметной области помимо ER-диаграммы необходимо представить текстовое описание сущностей и их атрибутов, а также связей между сущностями. Текстовое описание может должно представлено в виде таблицы 4.

Рисунок 14 – Физическая схема БД

Таблица 4 – Описание сущностей БД

Наименование поля	**Назначение атрибута**	**Тип данных**	**Примечание**
lease_contract (договор аренды):			
lease_contract	номер договора аренды	целое число типа INT, принимает значения из диапазона [10000; 99999]	первичный ключ
inventory_number	инвентарный номер помещения	целое число типа INT, принимает значения из диапазона [1001000; 9009999]	внешний ключ, связывает с таблицей premise, кардинальность связи – M:0
tenant_id	номер арендатора	целое беззнаковое число типа INT	внешний ключ, связывает с таб-

Наименование поля	Назначение атрибута	Тип данных	Примечание
			лицей tenant, кардинальность связи – M:1
lease_start_date	хранит дату начала аренды помещения	значение, соответствующие маске ДД-ММ-ГГГГ, начальное значение не должно быть ранее даты 17.04.2012	
lease_term	хранит число дней, на которое арендуется помещение	целое число типа SMALLINT, минимальное значение срока аренды составляет 10 дней, а максимальное – 180	

Описание остальных сущностей выполняется аналогично и размещается далее в таблице после характеристики первой сущности.

Краткие теоретические сведения и методические указания к Заданию 2

Архитектура программного обеспечения – фундаментальная организация системы, воплощенная в ее компонентах, их отношения друг к другу и к окружающей среде, а также принципы, лежащие в основе ее проектирования и развития.

Архитектура включает:

– выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов;

– соединение выбранных элементов структуры в более крупные системы и их поведение;

– архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Для того, чтобы проектируемая система отвечала различным атрибутам качества, на практике часто используют различные архитектурные шаблоны (паттерны).

Наиболее распространенные архитектурные шаблоны:

– Многоуровневый шаблон;

– Клиент-серверный шаблон;
– Ведущий-ведомый;
– Каналы и фильтры;
– Шаблон посредника;
– Одноранговый шаблон;
– Шина событий;
– Модель-представление-контроллер;
– Доска;
– Интерпретатор.

Краткий обзор архитектурных шаблонов представлен по ссылке: https://medium.com/nuances-of-programming/краткий-обзор-10-популярных-архитектурных-шаблонов-приложений-81647be5c46f

Проектирование может осуществляться с различных точек зрения.

Все диаграммы, используемые в UML 2.x, могут быть разбиты на две группы (рисунок 15):

– диаграммы для моделирования структуры программной системы – **структурные диаграммы**, показывающие **статическую структуру** системы и ее частей на разных уровнях абстракции и реализации;

– диаграммы для моделирования поведения программной системы – **диаграммы поведения**, показывающие **динамическое поведение** объектов в системе, которое можно описать как серию изменений в системе с течением времени.

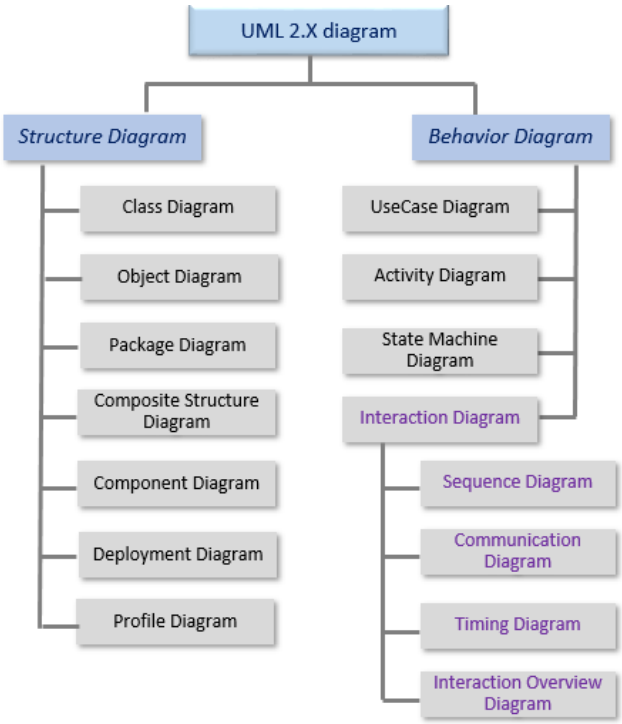


Рисунок 15 – Структурные диаграммы и диаграммы поведения (UML 2.x)

С позиции структурной организации в рамках лабораторной работы для проектирования архитектуры будут использоваться UML **диаграммы классов, компонентов, развертывания и пакетов.**

Каждому классу структурных диаграмм соответствует свой уровень абстракции в модели программной системы:

– логический уровень – диаграммы классов;
– уровень реализации – диаграммы компонент;
– уровень оборудования – диаграммы развертывания.

Диаграмма классов (рисунок 16). Диаграмма классов – это наиболее часто используемый тип диаграмм, которые создаются при моделировании объектно-ориентированных систем.

Диаграмма классов – это диаграмма, которая показывает набор классов, интерфейсов, кооперации и их связи.
Графически представляет собой набор вершин и связывающих их дуг.

На практике диаграммы классов применяют для моделирования статического представления системы (по большей части это моделирование словаря системы, кооперации или схем). Кроме того, диаграммы данного типа служат основой для целой группы взаимосвязанных диаграмм – диаграмм компонентов и диаграмм размещения.

Диаграммы классов важны не только для визуализации, специфицирования и документирования структурных моделей, но также для конструирования исполняемых систем посредством прямого и обратного проектирования.

Подробнее с примерами диаграмм классов можно ознакомиться в источниках [5] и [6].

В отчете по лабораторной работе необходимо представить диаграмму классов, отражающую используемый паттерн проектирования, и краткое описание задачи, которую решает данный паттерн.

Диаграмма компонентов (рисунок 17). С точки зрения реализации проектируемая система состоит из компонентов (представленных на диаграммах компонентов), распределенных по вычислительным узлам (представленным на диаграммах размещения).

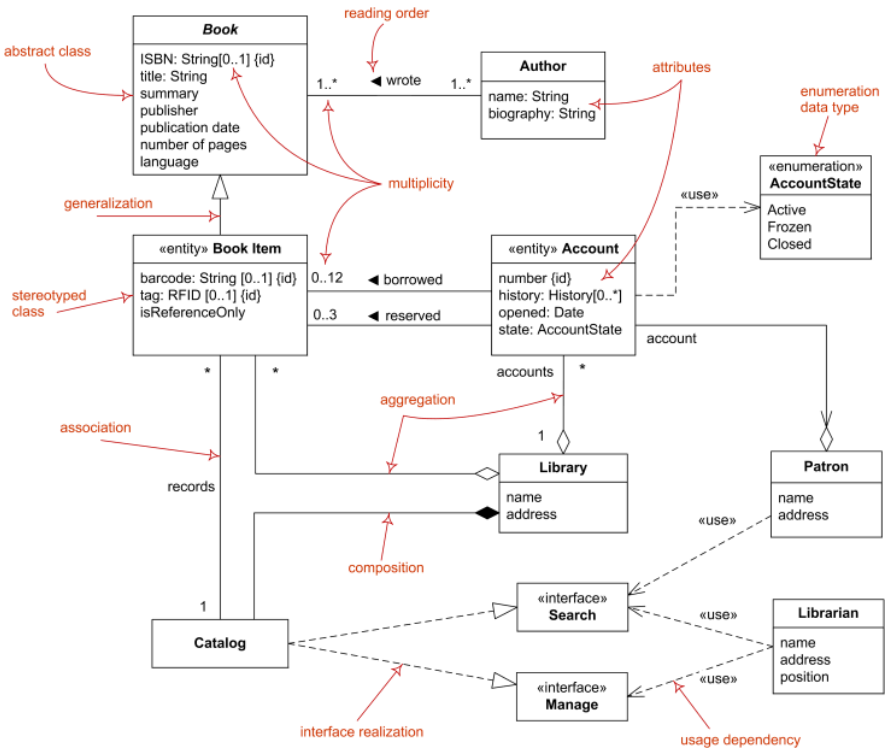


Рисунок 16 – Диаграмма классов

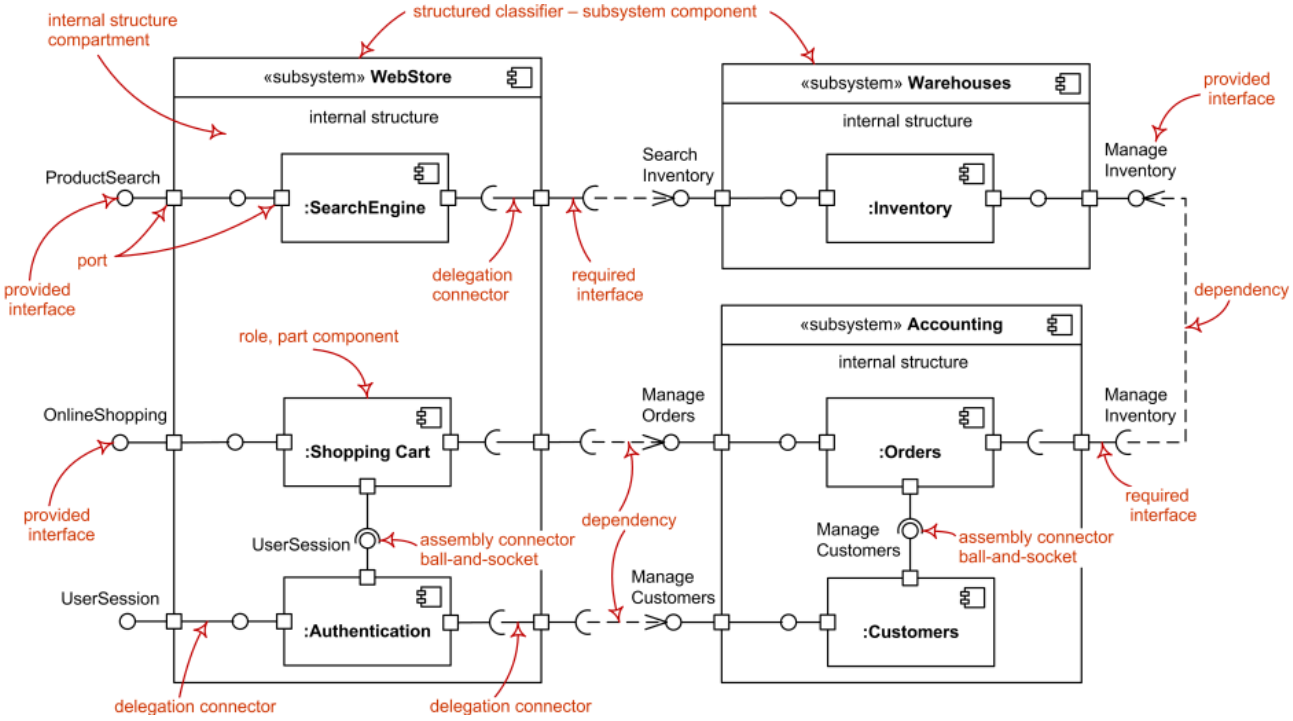


Рисунок 17 – Диаграмма компонентов

Диаграмма компонентов (component diagram) демонстрирует инкапсулированные классы и их интерфейсы, порты и внутренние структуры, состоящие из вложенных компонентов и коннекторов.
Диаграммы компонентов описывают статическое представление дизайна системы. Они важны при построении больших систем из мелких частей (UML отличает диаграмму составной структуры (composite structure diagram), применимую к любому классу, от компонентной диаграммы).

Одним из основных элементов данной диаграммы является компонент. Компоненты позволяют инкапсулировать части системы, чтобы уменьшить количество зависимостей, сделать их явными, а также повысить взаимозаменяемость и гибкость на случай, если система должна будет изменяться в будущем.

Хороший компонент наделен следующими характеристиками:
- инкапсулирует сервис с хорошо очерченным интерфейсом и границами;
- имеет внутреннюю структуру, которая допускает возможность ее описания;
- организует внешнее поведение, используя интерфейсы и порты в небольшом количестве;
- взаимодействует только через объявленные порты.

Более подробное описание других элементов диаграммы компонентов, а также примеры построения представлены в источнике [7, с. 206-220].

Диаграмма размещения (рисунок 18) – это один из двух видов диаграмм, используемых при моделировании физических аспектов объектно-ориентированной системы. Такая диаграмма представляет конфигурацию узлов, где производится обработка информации, и показывает, какие артефакты размещены на каждом узле.

Диаграмма размещения (deployment diagram) показывает конфигурацию узлов-процессоров, а также размещаемые на них компоненты. Диаграммы размещения дают статическое представление размещения архитектуры. Узлы, как правило, содержат один или несколько артефактов.

Диаграммы размещения используются для моделирования статического представления системы с точки зрения размещения. В основном под этим понимается моделирование топологии аппаратных средств, на которых работает система. По существу, диаграммы размещения – это просто диаграммы классов, сосредоточенные на системных узлах.

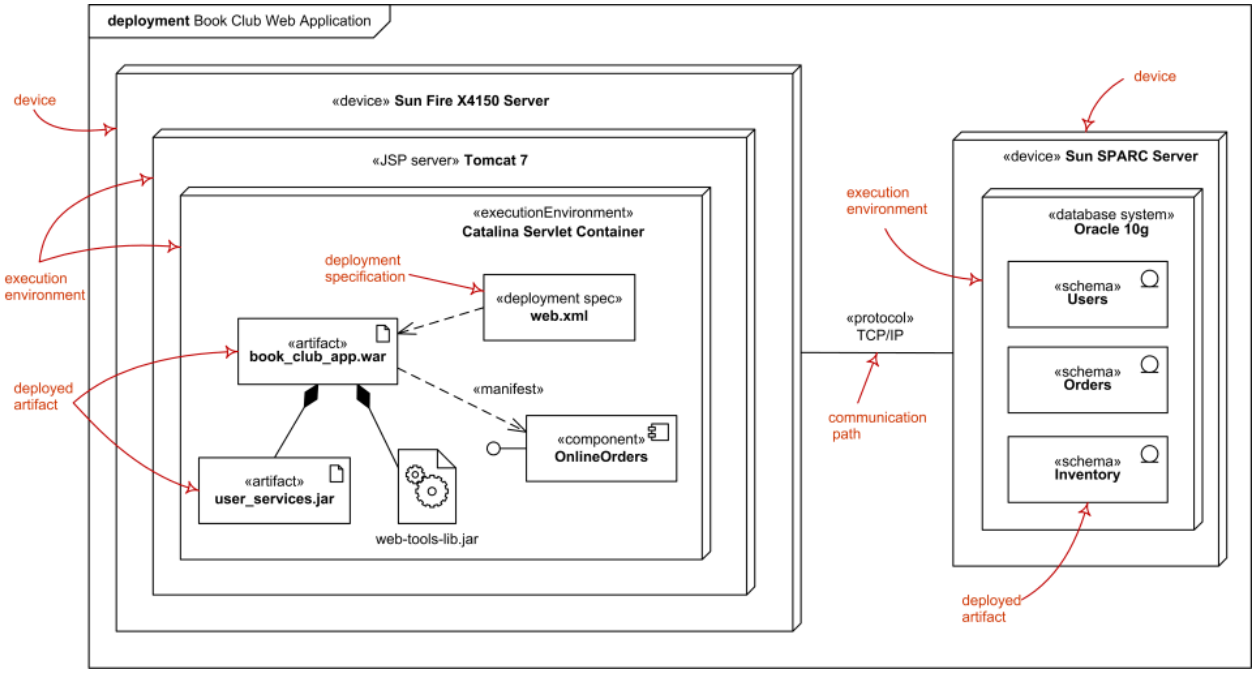


Рисунок 18 – Диаграмма размещения

Диаграммы размещения важны не только для визуализации, специфицирования и документирования встроенных, клиент-серверных и распределенных систем, но и для управления исполняемыми системами с использованием прямого и обратного проектирования.

Диаграмму размещения можно проигнорировать только в том случае, если разрабатываемое программное средство будет исполняться **исключительно** на одной машине, обращаясь при этом лишь к стандартным устройствам на этой же машине, управление которыми полностью возложено на ОС.

Подробнее с диаграммами размещения можно ознакомиться в источнике [7, с. 379-386, с. 427-438].

Диаграмма пакетов. В случае, если разрабатываемое программное средство будет иметь большое количество классов, интерфейсов, узлов, компонентов и других элементов, целесообразно построение диаграммы пакетов (рисунок 19).

Диаграмма пакетов (package diagram) показывает декомпозицию самой модели на организационные единицы и их зависимости

Основным элементом данной диаграммы является пакет – способ организации элементов модели в блоки, которыми можно распоряжаться как единым целым. Хорошо структурированный пакет объединяет семантически близкие элементы, которые имеют тенденцию изменяться совместно.

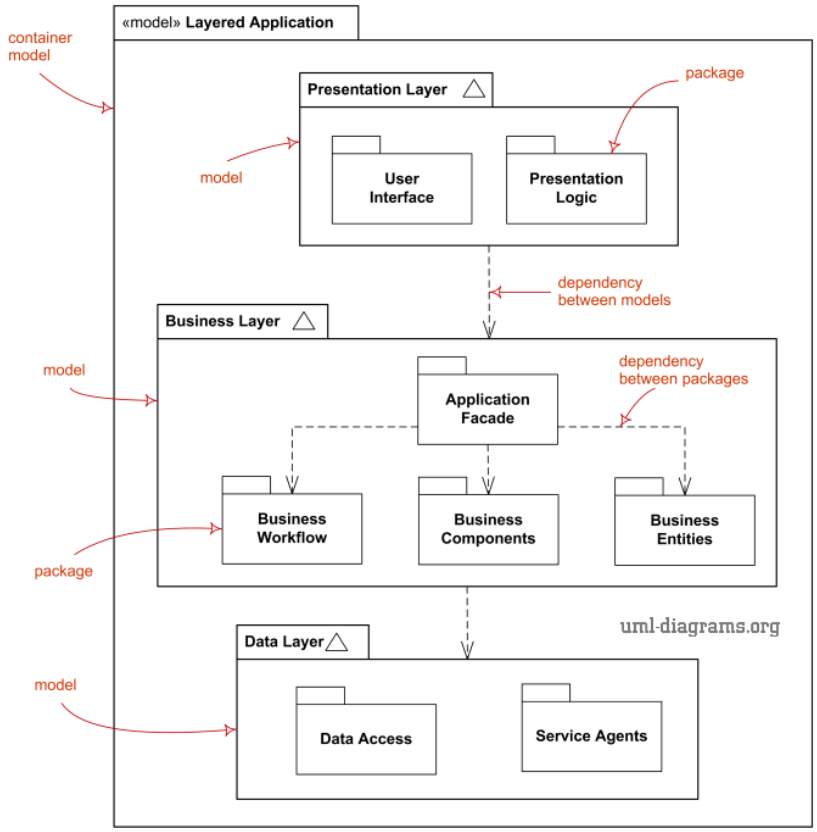


Рисунок 19 – Диаграмма пакетов

Пакеты характеризуются слабой связанностью и высокой согласованностью, причем доступ к содержимому пакета строго контролируется. Объединение частей модели в блоки упрощает процесс масштабирования системы в будущем.

Более детальное представление диаграмм компонентов приведено в источнике [7, с. 178-189].

Контрольные вопросы к лабораторной работе № 6

1 Что такое модель данных?

2 Какие аспекты рассматриваются при проектировании информационной модели предметной области?

3 Что такое нормализация и как привести отношение к 3 НФ?

4 Что понимают под архитектурой программной системы?

5 В чем заключается описание функциональной архитектуры программной системы?

6 Что такое архитектурный шаблон?

7 Как трактуется и представляется динамическое поведение объектов программной системы?

8 Как трактуется и представляется статическая структура программной системы и ее частей на разных уровнях абстракции и реализации?

Лабораторная работа № 7

Описание и разработка алгоритмов, реализующих бизнес-логику программного средства

Цели выполнения лабораторной работы:

- разработать алгоритмические и программные реализации бизнес-логики программного средства
- построить чертеж схемы программы

Задание	Этапы выполнения задания
1 Разработать алгоритмы, реализующие бизнес-логику программного средства, построить их схемы и привести фрагменты листинга кода	1 Разработать и представить текстовое описание алгоритмов, реализующих бизнес-логику 2 Построить схемы разработанных алгоритмов (схемы программы или схему работы системы) в соответствии с требованиями ЕСПД и ЕСКД 3 Привести фрагменты листинга кода программной реализации алгоритмов
2 Выполнить схему алгоритма в виде чертежа	1 Построить чертеж схемы программы либо схемы работы системы и оформить согласно требованиям ЕСПД и ЕСКД
3 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Описание схем алгоритмов 3 Чертежи схем алгоритмов 4 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

С точки зрения разработки программной системы **бизнес-логику** можно определить как совокупность правил, принципов, зависимостей поведения объектов предметной области.

В настоящее время существует множество различных подходов к описанию и реализации бизнес-логики. Однако выделяют три типовых решения[15]:

– **сценарий транзакции (функциональный подход)**. В этом случае проектируют функциональную структуру (иерархию функций) программной

[15] Данные решения детально рассмотрены в книге М. Фаулера «**Шаблоны корпоративных приложений**»

системы. Каждая функция системы реализует определенную операцию прикладной логики;

– **модель предметной области (объектный подход)**. Разрабатывается объектная модель предметной области. В этом случае бизнес-логику можно описывать посредством моделирования отношений и распределения ответственности между объектами.

– **модуль таблицы** (смешанный подход). Использует сочетание сценария транзакции и модели предметной области.

Неплохой пример упрощенной схемы выбора типового решения для реализации бизнес-логики представлен на рисунке 20:

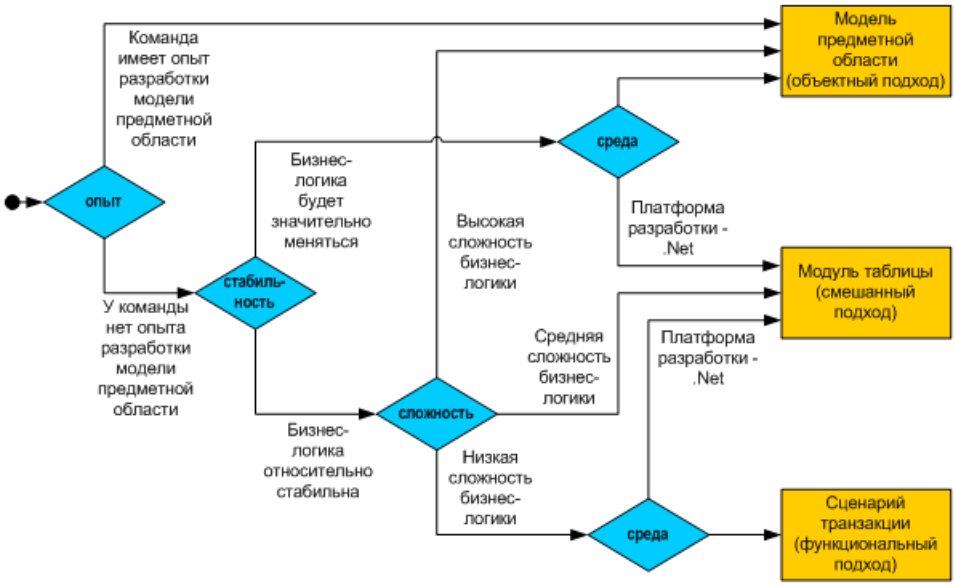


Рисунок 20 – Упрощенная схема выбора типового решения для реализации бизнес-логики[16]

На стадии анализа и моделирования программной системы бизнес-логика может описываться в виде:

– текста;

– концептуальных аналитических моделей предметной области;

– разнообразных алгоритмов;

– диаграмм UML – диаграмм деятельности, последовательности, состояний;

– моделей бизнес-процессов.

На стадии проектирования программной системы бизнес-логика реализуется в классах и методах классов, или процедур и функций.

Для детализации алгоритмов бизнес-логики будем использовать текстовое описание и схемы.

[16] Пример схемы выбора типового решения для реализации бизнес-логики взят из источника по ссылке https://software-testing.ru/library/testing/test-management/ 62-business-logic

Схема является универсальной, не зависящей от чего-либо, и отображает принцип работы определённого алгоритма, что даёт полное понимание происходящего процесса. Она описывает действие алгоритма так, как его можно реализовать на любом языке программирования.

Следует отметить, что схемы могут использоваться на различных уровнях детализации алгоритмов бизнес-логики.

Согласно ЕСПД, **схема** – графическое представление определения, анализа или метода решения задач, в котором используются символы для отображения операций, данных, потока, оборудования и т.д.

В ГОСТ 19.701-90 [8] выделены следующие типы схем:

– схема данных;
– схема программ;
– схема работы системы;
– схема взаимодействия программ;
– схема ресурсов системы.

Символы, использование которые допускается в такого рода схемах, представлены в таблице «Применение символов» ГОСТ 19.701-90 [8].

В рамках лабораторной работы описание разработанных алгоритмов следует представить в виде текста, а также в виде схем программ (отображают последовательность операций в программах).
Программную реализацию разработанных алгоритмов необходимо представить в виде небольшого фрагмента листинга кода.

Пример описания алгоритма, реализующего бизнес-логику программного средства

Алгоритм формирования заявки на обучение. Описание шагов алгоритма:

1) шаг 1 Пользователь авторизуется в системе. Вводит логин и пароль;
2) шаг 2 Осуществляется проверка логина и пароля. При успешной авторизации пользователю отображается список всех доступных курсов. В противном случае выдается сообщение об ошибке, и работа завершается;
3) шаг 3 При отображении списка доступных курсов пользователь выбирает интересующий курс и добавляет его в заявку;
4) шаг 4 Система проверяет, является ли выбранный пользователем курс новым. Если новый – система формирует заявку на обучение и выводит сообщение о назначении данного курса пользователю. В противном случае пользователю отобразится сообщение об ошибке.

Схема алгоритма формирования заявки на обучение представлена на рисунке 21.

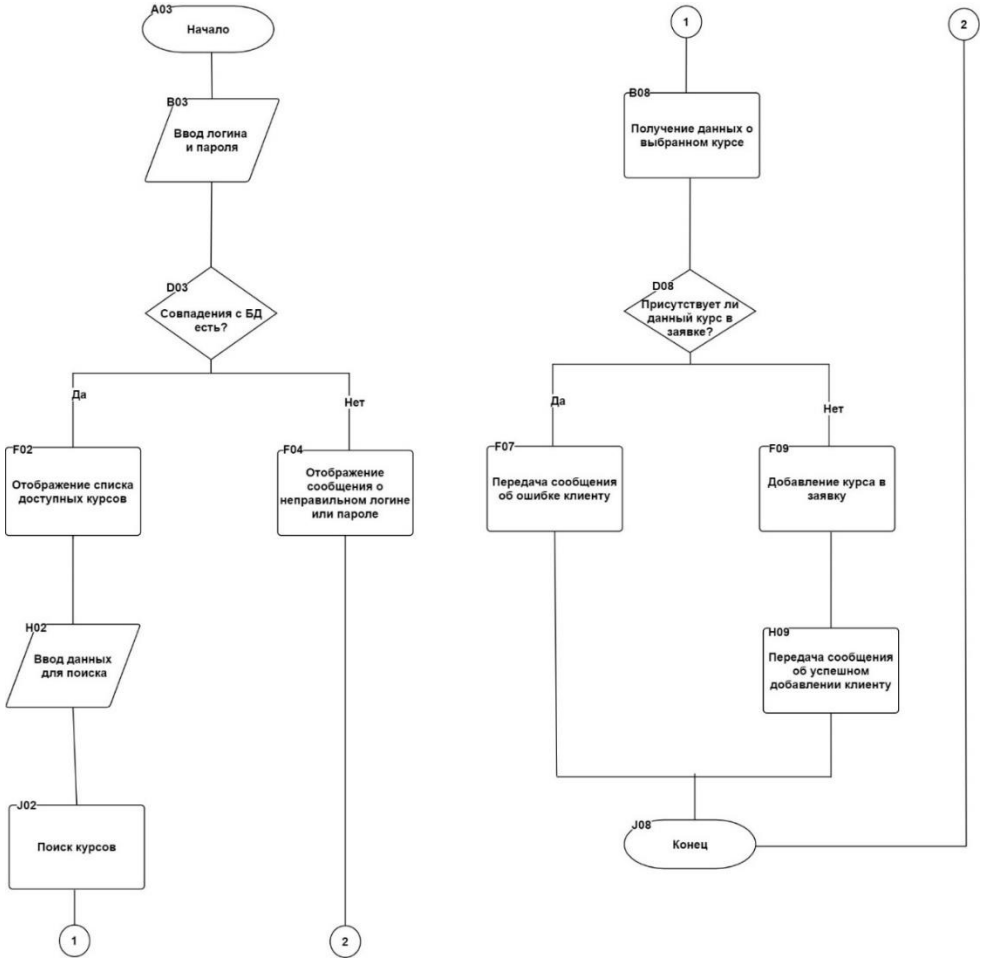


Рисунок 21 – Схема алгоритма формирования заявки на обучение

Для реализации алгоритма формирования заявки на обучение был создан отдельный метод класса *OrderTable*. Пользователь через форму обучающегося выбирает интересующий его курс. По нажатию кнопки Добавить данные о курсе передаются на сервер. При получении сообщения от пользователя, начинающегося с «*addToOrder*», вызывается метод *addToOrder класса OrderTable*, код которого представлен ниже:

```
public String addToOrder(String student_id, String course_name, String
course_format, String start_date){
    boolean isOrderHasEquals = false;
    boolean isUpdatesDone = false;
    try{
        String course_id = courseTable.getCourseParam(course_name,
course_format, start_date, "id");
        ResultSet rs = getUnacceptedOrder(student_id);
        if(!rs.next()){
            int done1 = stmt.executeUpdate("INSERT INTO ORDERS (ORD_ID,
ST_ID, STATUS) VALUES(ORDER_SEQ.nextval,"+student_id+", 0)");
            int done2 = stmt.executeUpdate("INSERT INTO STUDY_ORDER (ord_id,
st_proc_id) VALUES (ORDER_SEQ.currval,"+ quotate(course_id));
            if(done1 > 0 && done2 > 0 ){
                isUpdatesDone = true;
```

На сервере производятся все необходимые обработки исключительных ситуаций, которые могут возникнуть при добавлении курса в заявку с помощью оператора *try-catch*. С помощью цикла *if* делаются различные логические проверки. Для взаимодействия с базой данных используется метод *executeUpdate* который выполняет такие команды, как *INSERT*, *UPDATE*, *DELETE*, *CREATE TABLE*, *DROP TABLE*. В качестве результата возвращает количество строк, затронутых операцией (например, количество добавленных, измененных или удаленных строк), или 0, если ни одна строка не затронута операцией или если команда не изменяет содержимое таблицы.

Алгоритм обработки заявки администратором. Аналогично описать указанный алгоритм.

Краткие теоретические сведения и методические указания к Заданию 2

Чертежи разрабатываются с целью декомпозиции и пояснения сложных задач проектирования. Наименования и обозначения черчений должны соответствовать требованиям, установленным ГОСТ 2.102-68.

Пример схемы алгоритма обработки данных, оформленной как чертеж, представлен на рисунке 22.

При **построении чертежа** необходимо обратить внимание на следующие требования:

1 Каждый элемент схемы должен иметь ширину и длину равную ширине и длине ячейки сетки (подробнее о том, какими по размеру могут быть ячейки сетки можно узнать из ГОСТ 19.002-80 [9]).

2 Каждому элементу должны быть присвоены координаты согласно зоне, которую занимает данный блок (см. п.1.4. ГОСТ 19.002-80 [9]), при этом буква и цифра координаты вписывается в левом верхнем углу блока с отступом в 5 мм от края блока.

3 Изломы линий выполняются под углом в 90°.

Оформление основной надписи и прочих элементов осуществлять в соответствии с требованиями задания 2 лабораторной работы №2 (см. рисунок 7)[17].

[17] Тонкович, И.Н., **Разработка программного обеспечения: планирование, анализ и моделирование**: метод. пособие по дисциплине «Технологии проектирования сложных информационных систем» для студентов очной формы обучения учреждений высшего образования направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте) / И. Н. Тонкович, А. В. Шелест. – Репозиторий БГУИР, 2023. – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/50325.

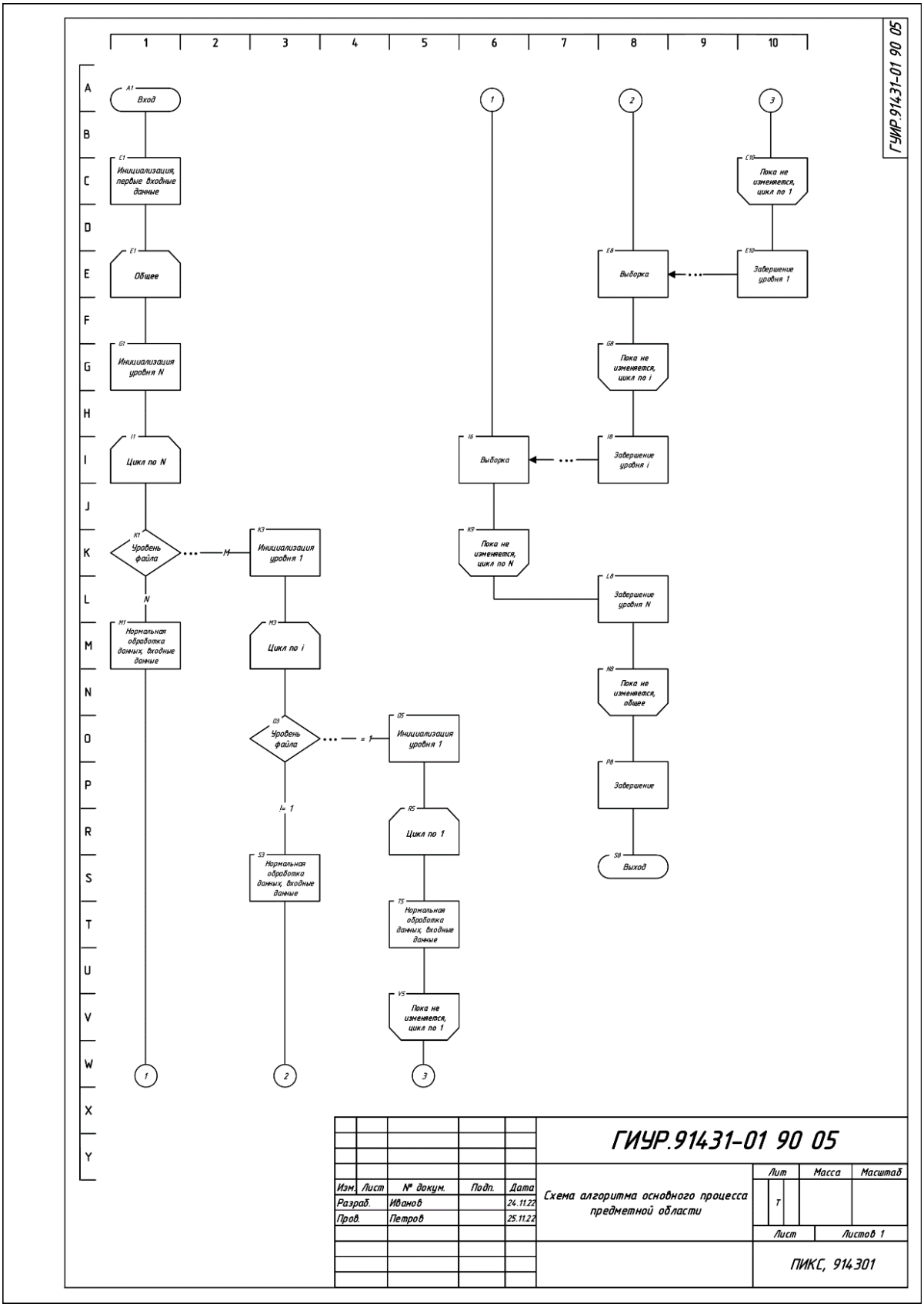


Рисунок 22 – Пример схемы алгоритма обработки данных

Контрольные вопросы к лабораторной работе № 7

1 Что понимают под бизнес-логикой программной системы и каковы ее основные компоненты?

2 Какие существуют типовые подходы к описанию и реализации бизнес-логики?

3 Как описывается бизнес-логика на стадии анализа и моделирования программной системы?

4 Как описывается бизнес-логика на стадии проектирования программной системы?

5 В чем заключаются основные концепции, используемые при разработке бизнес-логики?

6 Какое место занимает бизнес-логика в многоуровневой архитектуре программных систем?

7 Что такое схема алгоритма?

8 Какие выделяют типы схем алгоритма?

9 Каким образом описывается и документируется алгоритм функционирования программной системы?

10 Какие требования устанавливаются к выполнению и оформлению схемы алгоритма в виде чертежа?

Лабораторная работа № 8

Описание динамических аспектов поведения объектов системы

Цели выполнения лабораторной работы:

- описать динамические аспекты поведения сложных объектов программной системы

Задание	Этапы выполнения задания
1 Разработать и описать модели поведения сложных объектов программной системы **для конкретного варианта использования**	1 Средствами statechart диаграммы смоделировать состояния объектов и условия переходов между ними. Представить графическое и текстовое описание полученной модели поведения объектов 2 С помощью sequence диаграммы получить отражение во времени процесса обмена сообщениями между объектами. Представить графическое и текстовое описание полученной модели взаимодействия объектов
2 Сформировать отчет по лабораторной работе	1 Титульный лист (приложение А) 2 Результаты моделирования поведения объектов программной системы 3 Результаты моделирования взаимодействия объектов программной системы 4 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

Представление системы с определенной точки зрения (view point) называется **видом** (view). Вид также называют **моделью** системы с определенной точки зрения.

Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точку зрения и является объектом деятельности различных пользователей с конкретными интересами, ролями или задачами. Модель абстрагирует свойства, поведение и структуру системы.

Программная система может рассматриваться **со следующих точек зрения**:

- функциональных требований к системе (use case view);
- логической организации и поведения системы (logical design view);
- функционирования или работы процессов системы (process view);

– компонентной организации и поведения системы (implementation view или component view);

– требований к размещению компонентов на аппаратных ресурсах (deployment view).

В зависимости от точки зрения получают различные **виды или модели представления программной системы**:

- **Use case view** (представление вариантов использования или прецедентов) – предназначено для моделирования функциональных требований к системе;
- **Logical view** (логическое представление) – предназначено для моделирования логической структуры и поведения системы;
- **Component view** (компонентное представление) – предназначено для моделирования архитектуры системы на уровне компонентов;
- **Deployment view** (представление размещения или развертывания) – предназначено для моделирования развертывания компонентов системы на аппаратуре.

Иногда Component view и Deployment view объединяют в один вид, который называют **Implementation view**.

В рамках лабораторной работы ограничимся рассмотрением вида представления **Logical view в целях моделирования поведения программной системы –** модели поведения и взаимодействия объектов. Модели представления **Use case view, Logical view (диаграмма классов), Component view и Deployment view** были рассмотрены ранее.

Для моделирования **поведения объектов** программной системы широко используются следующие диаграммы:

– **диаграмма деятельности** (activity diagram) – моделируются последовательности действий, исполняемых объектами программной системы;

– **диаграмма состояний** (statechart diagram или state machine diagram) – моделируются состояния объектов программной системы, а также переходы между этими состояниями.

Для моделирования **взаимодействия объектов** программной системы используются диаграммы взаимодействия, включающие чаще всего два вида диаграмм:

– **диаграмму последовательности** (sequence diagram) – моделирует упорядоченное во времени взаимодействие объектов системы;

– **диаграмму взаимодействия** или **коммуникации** (collaboration или communication diagram) – моделирует общую схему взаимодействия объектов и роли объектов во взаимодействии.

Диаграммы деятельности[18] используются для моделирования поведе-

[18] Подробнее с примерами диаграмм деятедьности можно ознакомиться по ссылке https://www.uml-diagrams.org/activity-diagrams-examples.html

ния системы в рамках различных вариантов использования или потоков управления, а также для моделирования бизнес-процессов.

Однако, применительно к бизнес-процессам, желательно выполнение каждого действия ассоциировать с конкретным подразделением компании, т.е. использовать дорожки, позволяющие указать зоны ответственности в рамках моделируемого бизнес-процесса. В качестве имен дорожек используются либо названия подразделений (департаментов) рассматриваемой компании, либо названия отдельных должностей сотрудников тех или иных подразделений.

Диаграммы деятельности также удобны и для описания поведения, включающего большое количество параллельных процессов. Не менее важная область их применения связана с моделированием бизнес-процессов при параллельном программировании, поскольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Принять во внимание при разработке диаграммы деятельности:

- Деятельность (действие) на диаграмме изображается **прямоугольником со скругленными сторонами (а не углами!!!).**
- С точки зрения топологии стрелок узлы **decision (ромб) и fork (утолщенная линия)** схожи (один входной поток и несколько выходных). Разница между ними в параллельности: на выходе **decision-узла** появляется только один поток (из нескольких возможных), а на выходе **fork-узла** – несколько параллельных потоков.
- Хорошо структурированная диаграмма деятельности сконцентрирована на описании одного аспекта (варианта использования) динамики системы. Должна содержать только те элементы, которые существенны для понимания этого аспекта.
- Сложные деятельности можно дополнительно детализировать, разбив на действия и изобразив «диаграмму в диаграмме».

Диаграмма состояний (рисунок 23)[19] предназначена для моделирования динамического поведения объектов модели, используя их состояния и переходы между этими состояниями.

Диаграмма состояний определяет все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий. Диаграммы состояний чаще всего используются для описания поведения отдельных объектов, а также спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, системы.

[19] Подробнее с примерами диаграмм состояний можно ознакомиться по ссылке https://www.uml-diagrams.org/state-machine-diagrams-examples.html

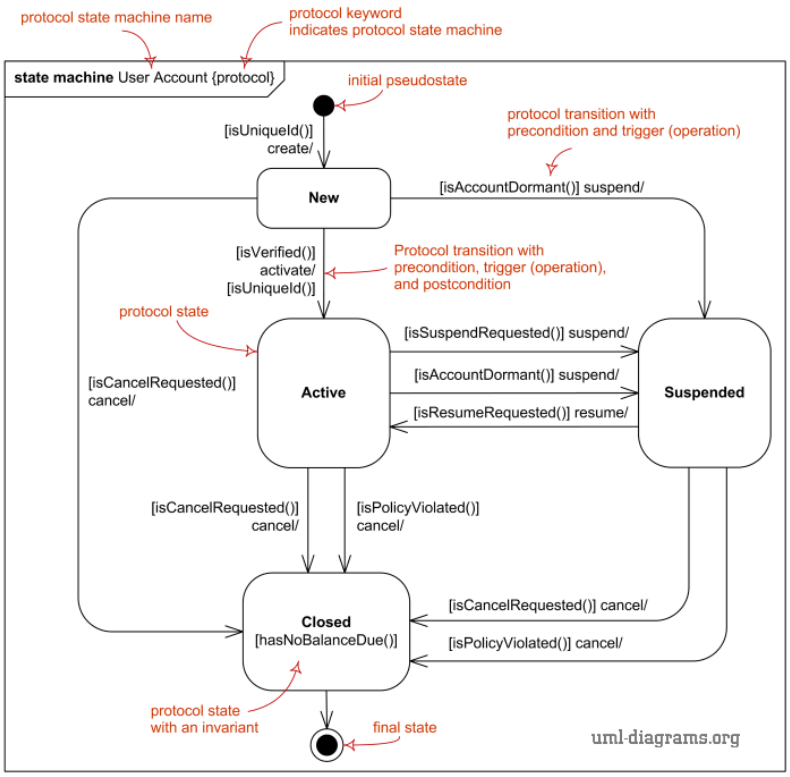


Рисунок 23 – Диаграмма состояний

Принять во внимание при разработке диаграммы состояний:

- Состояние на диаграмме изображается прямоугольником со скругленными вершинами (а не сторонами!!!).
- На диаграмме состояний decision-узлы не используем!!!
- **Диаграммы состояний не следует создавать для каждого класса**. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, то для него может потребоваться диаграмма состояний. **Диаграммы состояний создаются для описания объектов с высоким уровнем динамического поведения**.
- Из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние. Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной (неадекватной), либо ошибочной с точки зрения нотации языка UML.

При построении диаграммы необходимо выполнить требование отсутствия конфликтов у всех переходов, выходящих из одного и того же состояния. Наличие такого конфликта может служить признаком ошибки, либо параллельности или ветвления рассматриваемого процесса. Если параллельность по замыслу разработчика отсутствует, то следует ввести дополнительные сторожевые условия либо изменить существующие, чтобы исключить конфликт переходов. При наличии параллельности следует заменить конфликтующие переходы одним параллельным переходом типа ветвления.

Диаграммы взаимодействия описывают поведение взаимодействую-

щих групп объектов (в рамках варианта использования или некоторой операции класса). Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного потока событий варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Диаграммы последовательности и коммуникации, по сути, отображают одну и ту же информацию, но представляют ее с различных точек зрения. Подобно диаграммам последовательности кооперативные диаграммы отображают поток событий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы концентрируют внимание на связях между объектами. Большинство Case-средств позволяет после построения одной из диаграмм автоматически получить другую, а также выполнять синхронизацию этих диаграмм между собой.

В рамках лабораторной работы для моделирования **взаимодействия объектов** программного средства следует разработать только **диаграмму последовательности**.

Диаграмма последовательности (рисунок 24) [20] описывает поведение взаимодействующих объектов **в рамках реализации конкретного варианта использования**.
Как правило, ее используют для уточнения варианта использования, более детального описания логики сценариев использования.

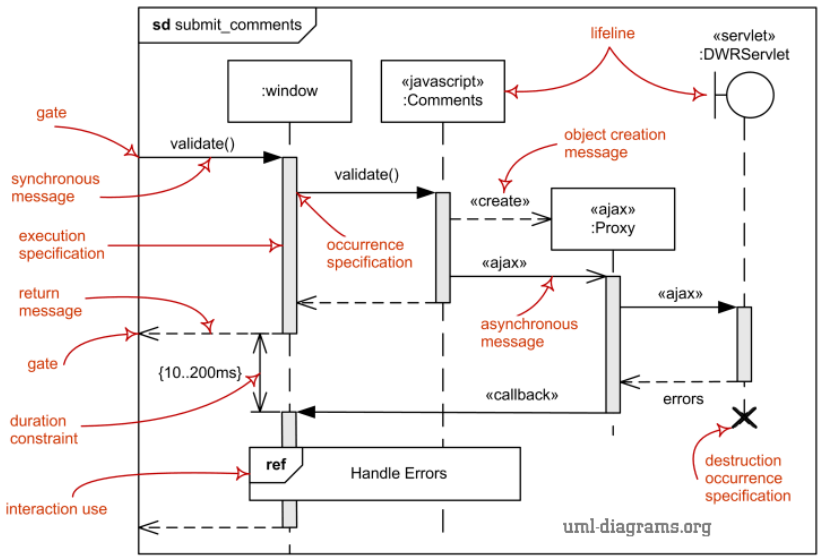


Рисунок 24 – Диаграмма последовательности

Для описания сообщений может быть использован шаблон, представленный таблицей 5.

[20] Подробнее с примерами диаграмм состояний можно ознакомиться по ссылке https://www.uml-diagrams.org/sequence-diagrams-examples.html

Таблица 5 – Шаблон описания сообщений

Объект-отправитель	Объект-получатель	Имя сообщения

Принять во внимание при разработке диаграммы последовательности:

- Диаграмма последовательности охватывает поведение объектов в рамках **только одного потока событий варианта использования**. Для отображения других вариантов развития событий следует построить несколько диаграмм.
- Диаграммы последовательности обязательно опираются на диаграмму классов. Объекты должны принадлежать классам, описанным в диаграмме классов. Если создаваемый объект не может принадлежать ни одному из существующих классов, следует доработать диаграмму классов. Поэтому в диаграммах последовательности нет необходимости создавать новые объекты, их достаточно просто перетащить в рабочую область текущей диаграммы из соответствующей диаграммы классов и определить имя экземпляра данного класса.
- Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Когда объекты визуализированы, приступают к спецификации сообщений.

Более детальное представление рассмотренных диаграмм приведено в источниках [7] и Фаулер М., UML. Основы, 3е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с., ил.

Контрольные вопросы к лабораторной работе № 8

1 В чем суть методологии объектно-ориентированного проектирования программных систем?

2 Какую парадигму проектирования и разработки программного обеспечения поддерживает UML?

3 Как моделируют динамические аспекты программной системы в UML?

4 Как моделируют взаимодействия объектов в UML?

5 В каких случаях целесообразно использование диаграммы состояний и для описания поведения каких компонентов системы используется?

6 Для чего применяется диаграмма последовательности и как ее построить?

Лабораторная работа № 9

Программная реализация клиентской части программного средства

Цели выполнения лабораторной работы:

- реализовать функциональность клиентской части программного средства

Задание	Этапы выполнения задания
1 Разработать клиентскую часть программного средства	1 Реализовать графический интерфейс программного средства и представить описание разработанных экранов 2 Реализовать необходимую функциональность клиентской части
2 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Ссылка на публичный репозиторий github с кодом клиентской части программного средства 3 Диаграмма вариантов использования 4 Карта переходов пользовательского интерфейса 5 Описание результатов выполнения клиентской части 6 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

В рамках лабораторной работы необходимо реализовать всю функциональность клиентской части ПС. Для этого необходимо:

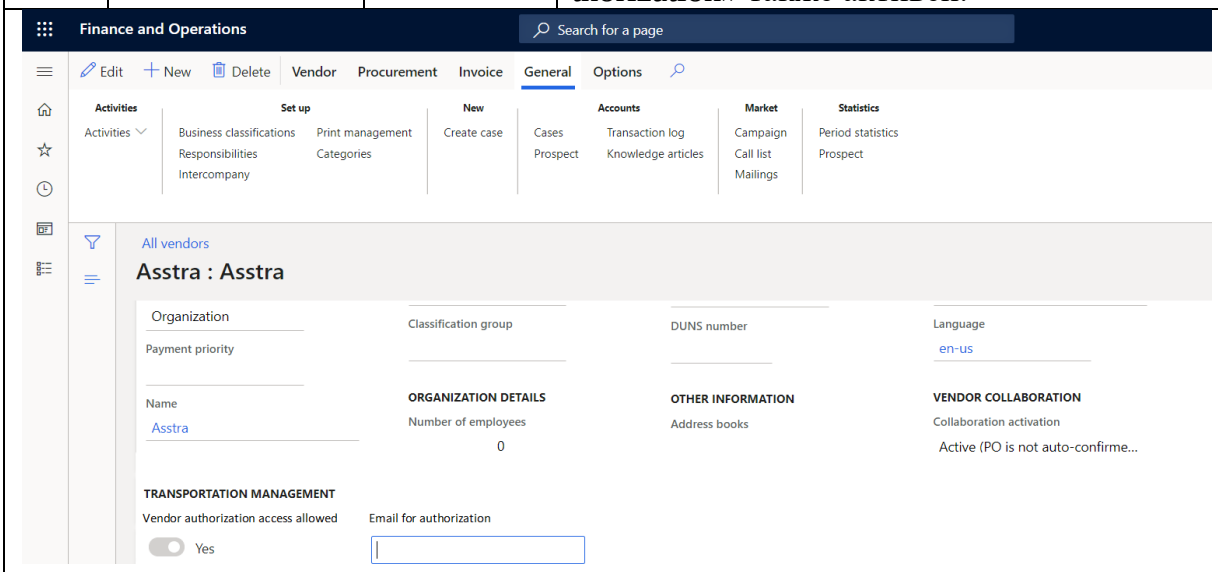
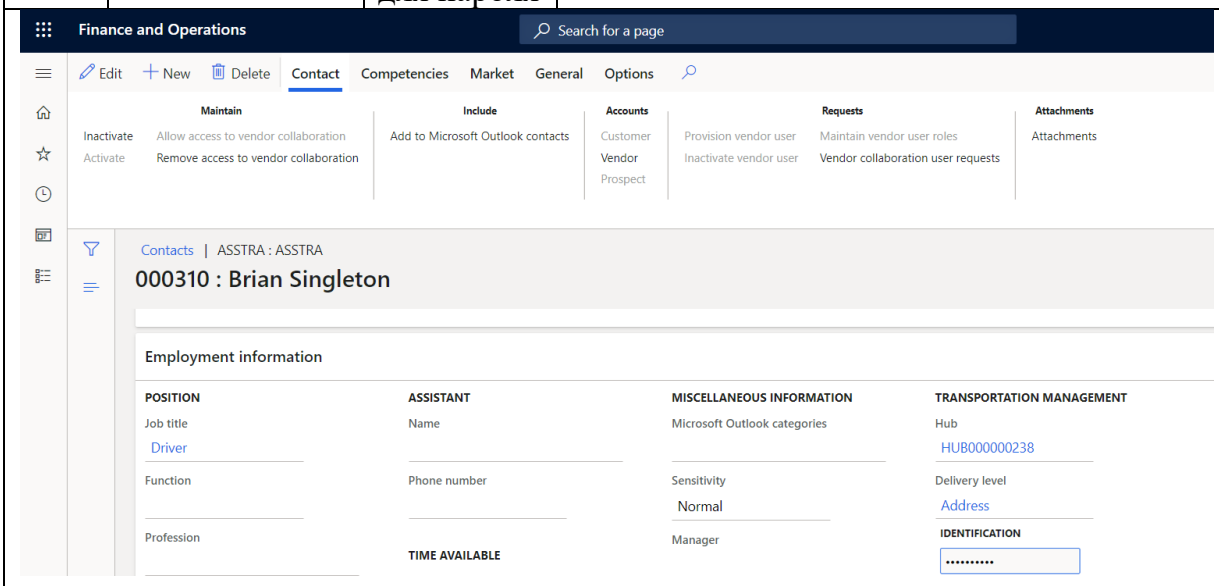
1 На основании разработанных в лабораторной работе №5 макетов и карты пользовательского интерфейса реализовать графический интерфейс программного средства.

2 Представить описание элементов всех экранов пользовательского интерфейса.

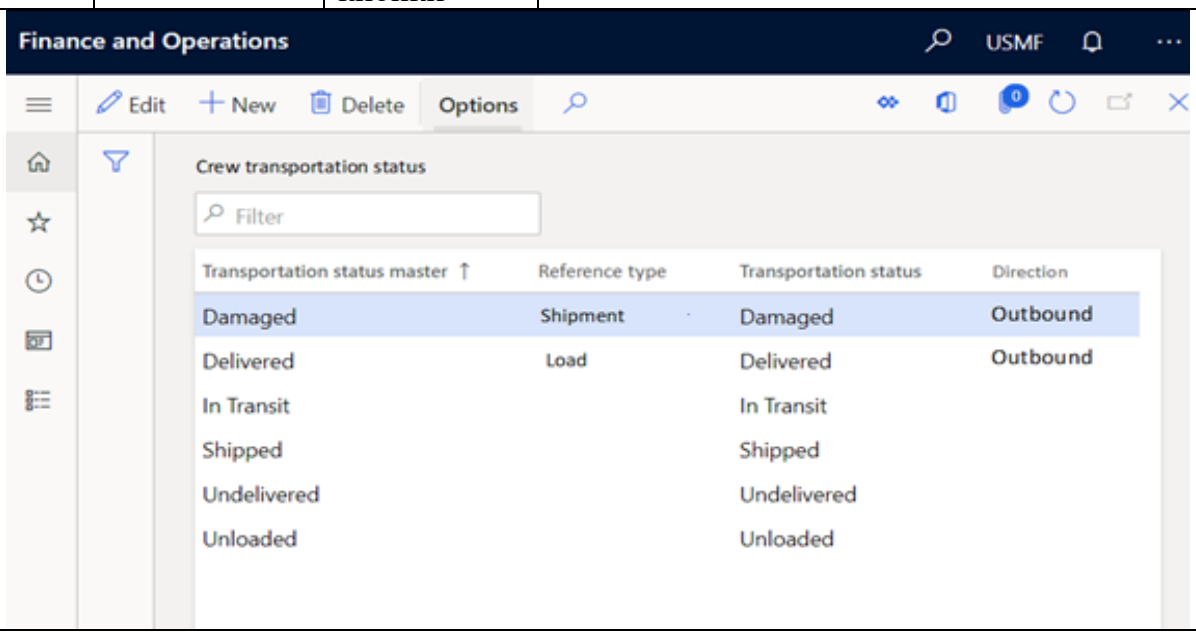
3 Используя выбранные для реализации программного средства технологии и компоненты (см. лабораторную работу №5) реализовать необходимую функциональность клиентской части, включая обработку исключительных ситуаций и введенных пользователем данных.

Для описания разработанных экранов может быть использован шаблон, представленный таблицей 6.

Таблица 6 – Шаблон описания реализованных элементов экранов пользовательского интерфейса

№ п/п	Реквизит (поле)	Значение	Комментарий
4.1	Добавление новой группы полей Transportation management		
4.1.1	Vendor authorization access allowed	Добавление ползунка	Активирует возможность авторизации в Power Apps. Help Text: Activates the account to be able to authorize in the Power Apps system
4.1.2	Email for authorization	Добавление поля	Поле для ручного ввода. Становится активным, если ползунок «Activate account for authorization» также активен.
			
4.2	Добавление поля Identification на форму Contact -> Employment Information -> Transportation management		
4.2.1	Identification	Добавление поля для пароля	
		

4.3	Создание новой формы «Crew transportation status»		
4.3.1	Transportation Status Master	Добавление нового поля	Справочник «Transportation Status Master»
4.3.2	Reference type	Добавление нового поля	Enum «TMSTransportRefType»
4.3.3	Transportation Status	Добавление нового поля	Справочник «Transportation Status», со значениями: ▪ Open ▪ In Transit ▪ Undelivered ▪ Unloaded ▪ Arrived ▪ Not Arrived ▪ Loaded ▪ Not Loaded ▪ Damaged ▪ Completed
4.3.4	Direction	Добавление нового поля	Справочник со значениями inbound и outbound
4.3.5	Edit	Добавление новой кнопки	
4.3.6	New	Добавление новой кнопки	
4.3.7	Delete	Добавление новой кнопки	



Контрольные вопросы к лабораторной работе № 9

1 Что такое клиентская часть?

2 Какие технологии используются для создания клиентских веб-приложений?

3 Каковы основные аспекты разработки клиентской части программной системы?

4 В чем различие между веб-приложением и веб-сайтом?

5 Какие существуют типы клиентских программных приложений?

6 Каким образом выполняются веб-приложения на клиентском компьютере?

7 Какие техники могут быть использованы для тестирования пользовательского интерфейса?

8 Для чего служат сценарии тестирования пользовательского интерфейса?

9 В каких случаях стоит использовать ручное тестирование пользовательского интерфейса вместо автоматизированного и наоборот?

10 В чем заключается отличие тестирования графического интерфейса (GUI testing) от тестирования удобства использования (Usability testing)?

Лабораторная работа № 10

Программная реализация серверной части программного средства

Цели выполнения лабораторной работы:

- реализовать функциональность серверной части программного средства в архитектурном стиле REST API с использованием принципов SOLID, DRY, KISS и шаблона DI

Задание	Этапы выполнения задания
1 Разработать серверную часть программного средства	1 Реализовать функциональность серверной части программного средства в архитектурном стиле REST API с использованием SOLID, DRY, KISS, шаблона DI 2 Составить документацию к разработанному API
2 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Ссылка на публичный репозиторий github с кодом клиентской части программного средства 3 UML диаграммы вариантов использования, классов, развертывания 4 Примеры кода, подтверждающих реализацию выбранных принципов и шаблонов программирования 5 Документация к разработанному API 6 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

REST – метод взаимодействия компонентов приложения с использованием протокола HTTP для вызова процедуры. При этом необходимые данные передаются в качестве параметров запроса. Этот способ является альтернативой более сложным методам, таким как SOAP, CORBA и RPC.

Веб-сервисы REST (рисунок 25) являются веб-сервисами, реализуемые с использованием протокола HTTP и принципов REST. Как правило, веб-сервис RESTful определяет URL основного ресурса, поддерживаемые MIME-типы представления/ответа и операции.

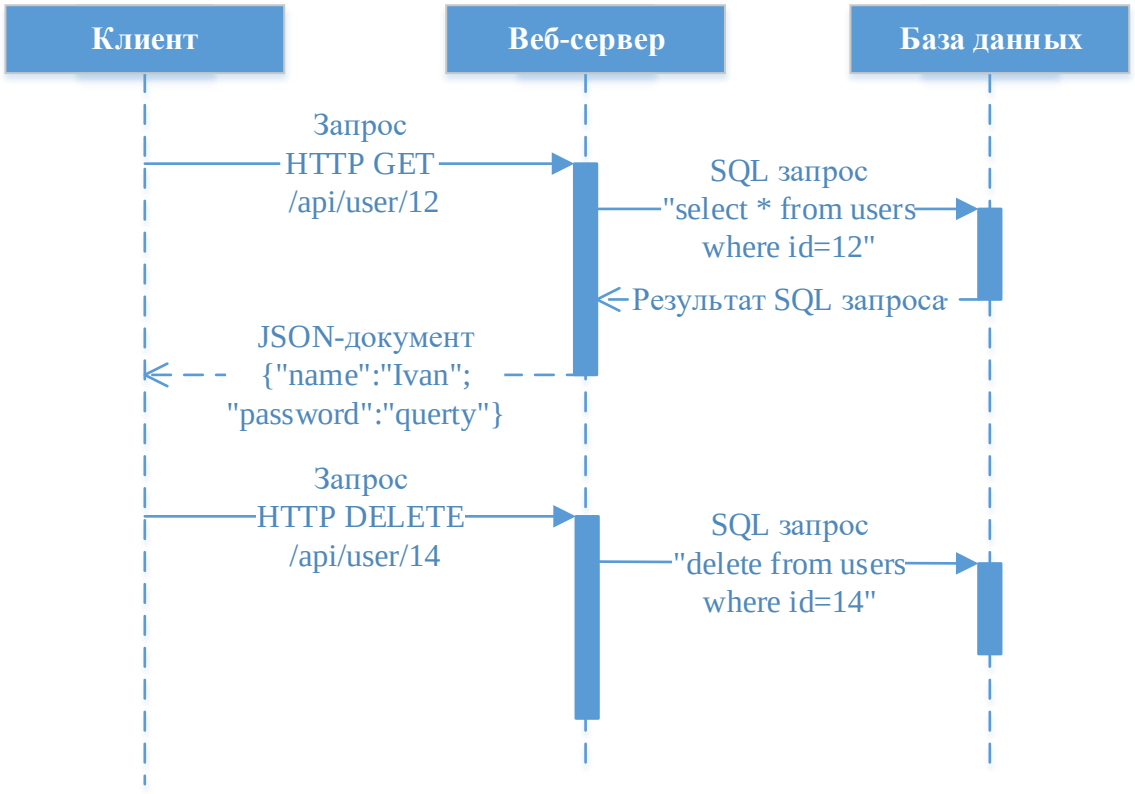


Рисунок 25 – REST веб-сервис

Пример запроса к REST веб-сервису

```
GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8
```

Пример ответа на запрос к REST веб-сервису

```
HTTP/1.1 200 OK
Content-Type: text/xml;
charset=utf-8;
Content-Length:nnn;
<?xmlversion="1.0"?>
<s:Quote xmlns:s="http://example.org/stock-service">
<s:TickerSymbol>IBM</s:TickerSymbol>
<s:StockPrice>45.25</s:StockPrice>
</s:Quote>
```

Ресурс может являться практически любым понятным и значимым адресуемым объектом.

Представление ресурса – это обычно документ, отражающий текущее или требуемое состояние ресурса. Ресурсы обычно представляются документами в форматах XML или JSON.

JSON (JavaScript object notation) – эффективный текстовый формат кодирования данных, обеспечивающий быстрый обмен небольшими объемами данных между клиентскими браузерами и веб-службами с поддержкой AJAX.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

1 Набор пар ключ: значение. В различных языках это реализовано как объект, запись, структура, словарь, хэш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка, значением – любая форма.

2 Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

REST и горизонтальный подход. Стратегия, опирающаяся на горизонтальный подход к протоколам, наиболее радикальна. Термин «горизонтальный» означает в данном случае сохранение существующего уровня, без выстраивания уровней поверх него. Предполагается отказаться от разработки новых протоколов, а использовать несколько хорошо проверенных, считая, что для работы с объектами вполне достаточно уметь выполнять четыре типа действий: создание (Creation), восстановление (Retrieval), изменение (Update) и уничтожение (Destruction). Из этих действий получается так называемый «шаблон проектирования» CRUD. Протокол Hypertext Transfer Protocol определяет методы GET/PUT/POST/DELETE (таблица 7), которые и реализуют шаблон CRUD. Абревиатура (шаблон) CRUD обозначает перечень основных операций с объектом: create (создать), read (считать, загрузить), update (обновить, изменить, отредактировать) и delete (удалить).

Таблица 7 – HTTP-методы в REST веб-сервисах

HTTP-метод	CRUD операция
GET	Read
POST	Create (иногда используется для операций update, delete)
PUT	Create (иногда используется для операции update)
DELETE	Delete

Разработчики SOAP веб-сервисов создают свой собственных перечень имен существительных и глаголов (например, getUsers(), savePurchaseOrder(...)) для обозначения операций веб-сервиса. В этом смысле SOAP реализуют сервисный принцип работы, в соответствии с которым главную роль при взаимодействии со службами играют их методы.

В основе архитектуры REST веб-сервисов лежит ресурсный (объектный) подход.

REST веб-сервисы разрабатываются согласно следующим принципам:

1 Возвращайте любые данные по их идентификатору.

2 Use standard methods – используйте стандарты. Имеется в виду, экономьте свои силы и деньги заказчика, используйте стандартные методы HTTP.

3 Одни и те же данные можно вернуть в различных форматах. Например, в XML или JSON для последующей программной обработки.

4 Передача данных без сохранения состояния. При обращении к REST-сервису не учитываются результаты ранее выполненных операций. REST приложение не сохраняет никакого состояния сессии на стороне сервера. Вся информация, необходимая для выполнения запроса, передается в самом запросе.

Часто REST веб-сервисы реализуются с помощью инфраструктуры создания MVC веб-приложений.

Принципы SOLID. В упрощенном варианте означают, что когда при написании кода используется несколько принципов вместе, то это значительно облегчает дальнейшую поддержку и развитие программы. Полностью акроним расшифровывается следующим образом:

1 **S**ingle responsibility principle – принцип единственной обязанности/ответственности (на каждый класс должна быть возложена одна единственная обязанность).

2 **O**pen/closed principle – принцип открытости/закрытости (программные сущности должны быть закрыты для изменения, но открыты для расширения);

3 **L**iskov substitution principle – принцип подстановки Барбары Лисков (функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом; подклассы не могут замещать поведения базовых классов; подтипы должны дополнять базовые типы).

4 **I**nterface segregation principle – принцип разделения интерфейса (много специализированных интерфейсов лучше, чем один универсальный).

5 **D**ependency inversion principle – принцип инверсии зависимостей (зависимости внутри системы строятся на основе абстракций; модули верхнего уровня не зависят от модулей нижнего уровня; абстракции не должны зависеть от деталей; детали должны зависеть от абстракций).

Пример реализации принципов SOLID представлен в источнике [11].

DRY (Don’t Repeat Youself). Этот принцип заключается в том, что нужно избегать повторений одного и того же кода. Лучше использовать универсальные свойства и функции.

Пример реализации принципа DRY на языке JavaScript

```
let chips = ['corn', 'pita', 'potato', 'tortilla'];

  // код, не соответствующий принципу DRY
  console.log(chips[0]);
  console.log(chips[1]);
  console.log(chips[2]);
```

```
// код, соответствующий принципу DRY
for (let i = 0; i < chips.length; i++) {
  console.log(chips[i]);
}
```

KISS (Keep It Simple, Stupid). Смысл этого принципа программирования заключается в том, что стоит делать максимально простую и понятную архитектуру, применять шаблоны проектирования.

Пример реализации принципа KISS

```
// код, не соответствующий принципу KISS
public String weekday1(int dayOfWeek)
{

    switch (dayOfWeek)
    {
       case 1: return "Monday";
       case 2: return "Tuesday";
       case 3: return "Wednesday";
       case 4: return "Thursday";
       case 5: return "Friday";
       case 6: return "Saturday";
       case 7: return "Sunday";
       default: throw new IllegalArgumentException("dayOfWeek must be in
range 1..7");
}
}
// код, соответствующий принципу KISS
public String weekday2(int dayOfWeek)
{
    if ((dayOfWeek < 1) || (dayOfWeek > 7))
        throw new IllegalArgumentException("dayOfWeek must be in range
1..7");

    final String[] weekdays = {
        "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Sat-
urday", "Sunday"};
    return weekdays[dayOfWeek-1];
}
```

Внедрение зависимости (англ. dependency injection, далее шаблон DI) – шаблон проектирования, суть которого заключается в следующем: зависимый объект (или функция) получает другой объект (или функцию), от которого он зависит. Данный шаблон позволяет уменьшать степень связанности программных компонентов. Другими словами, он дает возможность сократить риск возникновения непредвиденных изменений в программных элементах ввиду внесения изменений в другие части программы (например, при изменении провайдера данных необходимо изменить бизнес-логику программы). На текущий момент данный шаблон получил широкое распространение среди разработчиков программного обеспечения, т.к. позволяет создавать качественный программный код и строить более эффективные программные архитектуры.

Согласно требованиям заказчика, необходимо разработать REST-API сервис, внутренняя архитектура которого представлена на рисунке 26.

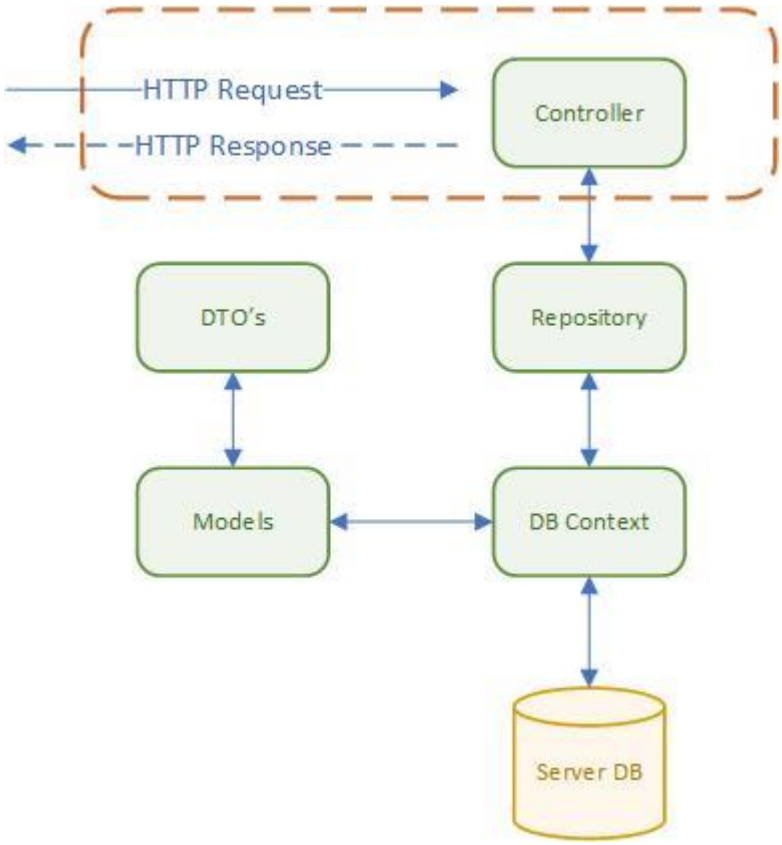


Рисунок 26 – Архитектура REST-API сервис

Фрагмент диаграммы классов в соответствии с выбранной архитектурой представлен на рисунке 27.

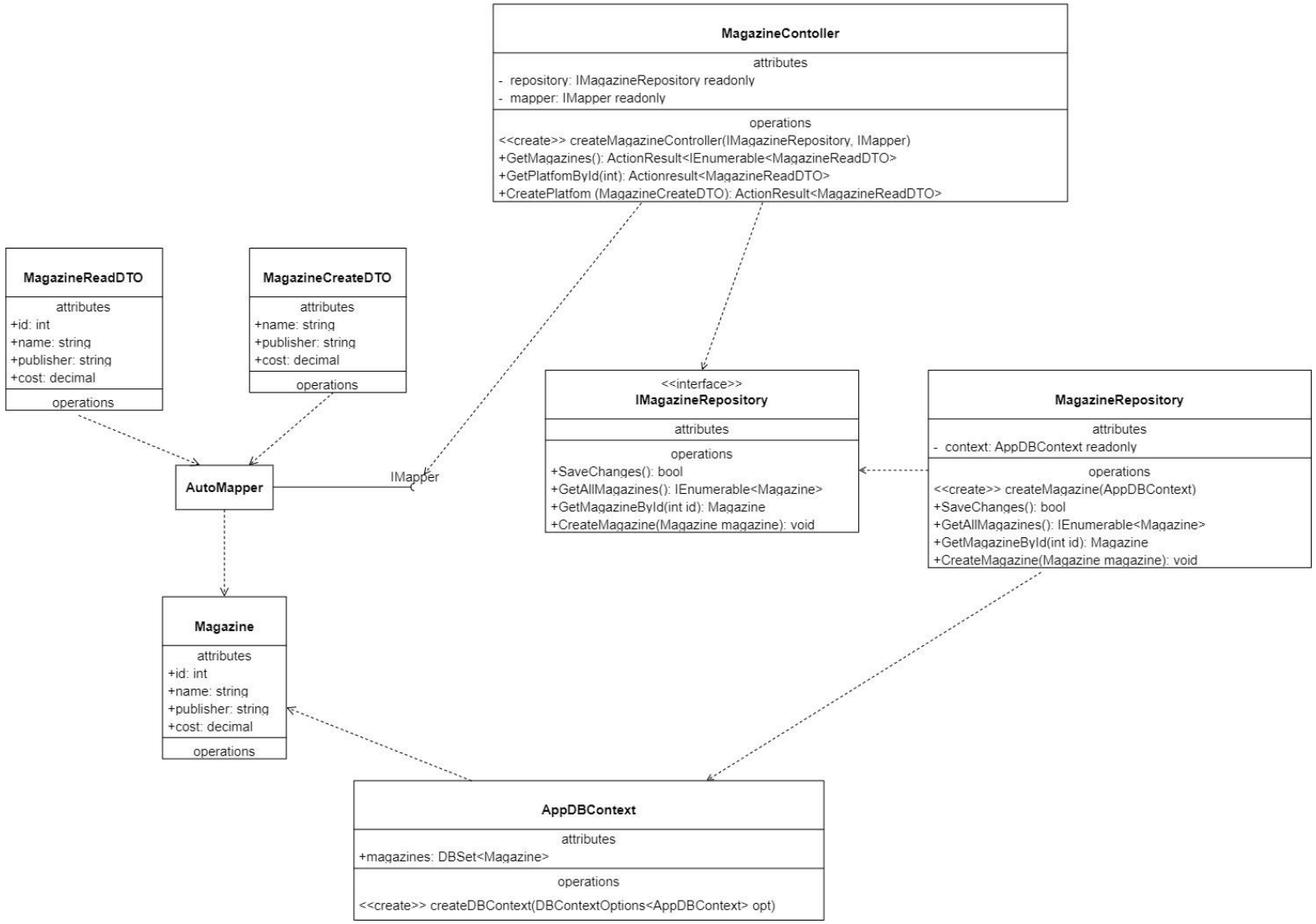


Рисунок 27 – Фрагмент диаграммы классов сервиса

Программный код, реализующий диаграмму классов, представленную на рисунке 28, размещен ниже.

```
public class Magazine
{
	[Key]
	[Required]
	public int Id { get; set; }

	[Required]
	public string Name { get; set; }

	[Required]
	public string Publisher { get; set; }

	[Required]
	public decimal Cost { get; set; }
}

public class MagazineReadDTO
{
	public int Id { get; set; }
	public string Name { get; set; }
	public string Publisher { get; set; }
	public decimal Cost { get; set; }
}

public class MagazineCreateDTO
{
	[Required]
	public string Name { get; set; }
	[Required]
	public string Publisher { get; set; }
	[Required]
	public decimal Cost { get; set; }
}

public class AppDbContext : DbContext
{
	public AppDbContext(DbContextOptions<AppDbContext> opt) : base(opt)
	{

	}

	public DbSet<Magazine> magazines { get; set; }
}

public interface IMagazineRepository
{
	bool SaveChanges();

	IEnumerable<Magazine> GetAllMagazines();
	Magazine GetMagazineById(int id);
	void CreateMagazine(Magazine magazine);
}
```

```
public class MagazineRepository: IMagazineRepository
{
	private readonly AppDbContext _context;

	public MagazineRepository(AppDbContext context)
	{
		_context = context;
	}

	public void CreateMagazine(Magazine magazine)
	{
		if(magazine == null)
		{
			throw new ArgumentNullException(nameof(magazine));
		}

		_context.Magazines.Add(magazine);
	}

	public IEnumerable<Magazine> GetAllMagazines()
	{
		return _context.Magazines.ToList();
	}

	public Magazine GetMagazineById(int id)
	{
		return _context.Magazines.FirstOrDefault(m => m.Id == id);
	}

	public bool SaveChanges()
	{
		return (_context.SaveChanges() >= 0);
	}
}

[Route("api/[controller]")]
[ApiController]
public class MagazinesController : ControllerBase
{
	private readonly IMagazineRepository _repository;
	private readonly IMapper _mapper;

	public MagazinesController(
		IMagazineRepository repository,
		IMapper mapper
		)
	{
		_repository = repository;
		_mapper = mapper;
	}

	[HttpGet]
	public ActionResult<IEnumerable<MagazineReadDTO>> GetMagazines()
	{
		var magazineItem = _repository.GetAllMagazines();

	return Ok(_mapper.Map<IEnumerable<MagazineReadDTO>>(magazineItem));
	}

	[HttpGet("{id}", Name = "GetMagazineById")]
	public ActionResult<MagazineReadDTO> GetMagazineById(int id)
	{
		var magazineItem = _repository.GetMagazineById(id);
```

```
            if(magazineItem != null)
            {
                  return Ok(_mapper.Map<MagazineReadDTO>(magazineItem));
            }

            return NotFound();
      }

      [HttpPost]
      public async Task<ActionResult<MagazineReadDTO>> CreateMagazine(
            MagazineCreateDTO magazineCreateDTO)
      {
            var magazineModel = _mapper.Map<Magazine>(magazineCreateDTO);
            repository.CreateMagazine(magazineModel);
            repository.SaveChanges();

            var magazineReadDTO = _mapper.Map<magazineReadDTO>(maga-
      zineModel);

            return CreatedAtRoute(nameof(GetMagazineById),
                        new {Id = magazineReadDTO.Id}, magazineReadDTO);
      }
}
```

Вышепредставленный листинг кода содержит пример реализации:

– принципа единственной ответственности (например, класс AppDBContext);
– принципа DRY (например, метод SaveChanges());
– принципа KISS (например, метод CreateMagazine());
– внедрения зависимости (например, переменная private readonly IMagazineRepository _repository).

- В рамках лабораторной работы необходимо реализовать функциональность серверной части программного средства в виде REST API[21]. Основная информацию по проектированию API представлена в источнике [12].
- Результат выполнения лабораторной работы – программный код, соответствующий принципам SOLID, DRY, KISS, а также использующий шаблон DI. В тексте отчета необходимо привести примеры кода с указанием того, какой именно принцип был реализован. Также необходимо представить UML диаграммы вариантов использования, классов и развертывания.
- В отчете необходимо представить документацию к разработанному API в виде таблицы[22], шаблон которой представляет таблица 8.

[21] Тип *API* можно изменить. Если будет выбран другой тип *API* нужно привести обоснование выбора выбранного вида *API*

[22] Пример описания был взят из источника по ссылке: *https://github.com/Medium/medium-api-docs#2-authentication*

Таблица 8 – Описание запросов и ответов UC-8

Параметр	Описание
Вариант использования	UC-8 Создать публикацию в профиле авторизованного пользователя
URL	`POST https://api.medium.com/v1/users/{{authorId}}/posts`

Пример запроса	`POST /v1/users/5303d74c64f66366f00cb9b2a94f3251bf5/posts HTTP/1.1`
	`Host: api.medium.com`
	`Authorization: Bearer 181d415f34379af07b2c11d144dfbe35d`
	`Content-Type: application/json`
	`Accept: application/json`
	`Accept-Charset: utf-8`
	
	`{`
	`  "title": "Liverpool FC",`
	`  "contentFormat": "html",`
	`  "content": "<h1>Liverpool FC</h1><p>You'll never walk alone.</p>",`
	`  "canonicalUrl": "http://jamietalbot.com/posts/liverpool-fc",`
	`  "tags": ["football", "sport", "Liverpool"],`
	`  "publishStatus": "public"`
	`}`

Пример ответа	`HTTP/1.1 201 OK`
	`Content-Type: application/json; charset=utf-8`
	
	`{`
	`  "data": {`
	`  "id": "e6f36a",`
	`  "title": "Liverpool FC",`
	`  "authorId": "5303d74c64f66366f00cb9b2a94f3251bf5",`
	`  "tags": ["football", "sport", "Liverpool"],`
	`  "url": "https://medium.com/@majelbstoat/liverpool-fc-e6f36a",`
	`  "canonicalUrl": "http://jamietalbot.com/posts/liverpool-fc",`
	`  "publishStatus": "public",`
	`  "publishedAt": 1442286338435,`
	`  "license": "all-rights-reserved",`
	`  "licenseUrl": "https://medium.com/policy/9db0094a1e0f"`
	`  }`
	`}`

Тип и описание параметров запроса и ответа на запрос представлены в таблицах 9 и 10.

Таблица 9 – Тип и описание параметров запроса (UC-8)

Параметр	Тип	Обязательный	Описание
title	string	да	Заголовок публикации. Используется для SEO. Отображается в случае представления публикаций в виде списка (в фактической публикации заголовок не отображается; чтобы заголовок был отображен в фактической публикации, необходимо указать его в поле content). Заголовки, длина которых больше 100 символов, будут игнорироваться. В этом случае заголовок будет сгенерирован из текста публикации при ее публикации
contentFormat	string	да	Формат поля content. Допустимы два возможных типа: html, markdown
content	string	да	Тело публикации семантически валидный фрагмент HTML или Markdown
tags	string array	нет	Теги для классификации публикаций. Использоваться будут только первые три тега. Теги, длина которых более 25 символов, будут игнорироваться
canonicalUrl	string	нет	URL исходной публикации, если изначально она была опубликована в другом месте
publishStatus	enum	нет	Статус публикации. Допустимые значения: public, draft, unlisted. Значение по умолчанию: public
license	enum	нет	Лицензия публикации. Допустимые значения: all-rights-reserved, cc-40-by, cc-40-by-sa, cc-40-by-nd, cc-40-by-nc, cc-40-by-nc-nd, cc-40-by-nc-sa, cc-40-zero, public-domain. Значение по умолчанию: all-rights-reserved
notifyFollowers	bool	нет	Параметр указывает, необходимо ли уведомлять подписчиков о публикации данной статьи

Таблица 10 – Описание и тип параметров ответа на запрос (UC-8)

Поле	Тип	Описание
id	string	Уникальный идентификатор публикации
title	string	Заголовок публикации
authorId	string	Уникальный идентификатор пользователя, опубликовавшего статью
tags	string array	Теги публикации
url	string	URL публикации
canonicalUrl	string	URL исходной публикации. Если данное поле не было специфицировано в процессе создания публикации, то оно будет отсутствовать
publishStatus	string	Статус публикации
publishedAt	timestamp	Дата публикации. Если была создана публикация со статусом draft, это поле не будет отображаться
license	enum	Лицензия публикации
licenseUrl	string	URL лицензии публикации

Перечень возможных ошибок, которые могут возникнуть в процессе выполнения запроса, представлен в таблице 11.

Таблица 11 – Перечень возможных ошибок (UC-8)

Код ошибки	Описание
400 Bad Request	Обязательные поля не действительны или не указаны
401 Unauthorized	Маркер доступа (access token) не действителен или был отозван
403 Forbidden	Пользователь не имеет прав на публикацию или значение поля authorId в пути запроса указывает на неправильно/несуществующего пользователя

Аналогично следует описать функции, реализующие все варианты использования.

При выполнении лабораторной работы можно использовать сторонние библиотеки, фреймворки, шаблоны и прочие дополнительные элементы. Если были использованы сторонние компоненты, на них должна быть приведена ссылка в тексте отчета по лабораторной работе в следующем виде: *<название источника>:<URL>*.
Весь код серверной части, необходимо разместить в публичном репозитории.

Контрольные вопросы к лабораторной работе № 10

1 Что такое REST?

2 Для чего используют RESTful API и как работает RESTful API?

3 Каковы преимущества и недостатки REST API?

4 Как используется HTTP при создании REST API?

5 Какова структура запроса REST API от клиента к серверу?

6 Какова структура ответа после выполнения REST API запроса?

7 Какие методы HTTP поддерживаются REST и когда они используются?

8 Каковы принципы REST?

9 Что такое коды состояния HTTP?

10 Что такое ресурс в REST?

11 В каком формате передаются данные при взаимодействии с REST веб-сервисом?

12 Каким образом кодируются данные в формате JSON?

13 Каковы лучшие ресурсы для изучения REST API?

14 Что такое принципы SOLID и зачем они нужны?

15 Как применять принципы SOLID на практике?

16 Что означает принцип DRY?

17 Что лежит в основе принципа KISS?

18 В чем состоит концепция DI?

Лабораторная работа № 11

Разработка тест-кейсов и юнит-тестов для проверки работоспособности программного средства

Цели выполнения лабораторной работы:

- провести тестирование разработанного программного средства с использованием тест-кейсов
- провести тестирование разработанного программного средства с использованием UNIT-тестов

Задание	Этапы выполнения задания
1 Разработать тест-кейсы для проверки работоспособности программного средства	1 Разработать тест-кейсы для проверки уровня базовых пользовательских требований и оформить результаты тестирования
2 Разработать UNIT-тесты для проверки работоспособности программного средства	1 Разработать UNIT-тесты для проверки работоспособности кода серверной части программного средства и оформить результаты тестирования 2 Оценить покрытие кода тестами
3 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Ссылка на публичный репозиторий github 3 Тест-кейсы для проверки уровня базовых пользовательских требований 4 Перечень функциональных требований к программному средству 5 Результаты автоматизированного тестирования функциональности программного кода 6 Листинг кода тестирования функциональных требований 7 Оценка тестового покрытия кода 8 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

Тестирование (testing): Процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и для определения дефектов **[глоссарий ISTQB].**

Подробная классификация видов тестирования представлена в источнике [10].
В рамках лабораторной работы для проверки уровня базовых пользовательских требований будем использовать тестирование на основе тест-кейсов.

Тестирование на основе тест-кейсов – формализованный подход, в котором тестирование производится на основе заранее подготовленных тест-кейсов, наборов тест-кейсов и иной документации.

Тест-кейс – набор входных данных, условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

Общая структура тест-кейса включает может включать следующие структурные элементы:

- идентификатор;
- связанное с тест-кейсом требование;
- модуль и подмодуль приложения;
- заглавие тест-кейса;
- исходные данные, приготовления к тест-кейсу;
- шаги тест-кейса (сценарий тест-кейса);
- ожидаемые результаты по каждому шагу тест-кейса;
- полученные результаты по каждому шагу тест-кейса.

В рамках лабораторной работы в качестве структурных элементов тест-кейса будем рассматривать следующие элементы:

- Идентификатор тест-кейса;
- Заглавие тест-кейса;
- Шаги тест-кейса;
- Ожидаемый результат.

Идентификатор тест-кейса – это уникальное значение, которое позволяет однозначно отличить один тест-кейс от другого.

Например, идентификатор UC-5 показывает связанное с тест-кейсом базовое пользовательское требование, которое приведено в спецификации вариантов использования.

Заглавие тест-кейса необходимо для понимания основной идеи тест-кейса без обращения к его остальным атрибутам. Заглавие тест-кейса должно быть информативным и уникальным.

Шаги тест-кейса описывают последовательность действий, которые необходимо реализовать в процессе выполнения тест-кейса.

Ожидаемые результаты по каждому шагу тест-кейса описывают реакцию приложения на действия, описанные в поле «шаги тест-кейса». Номер

шага соответствует номеру результата. Результат тест-кейса может быть положительным (фактический результат совпадает с ожидаемым), отрицательным (фактический результат отличается от ожидаемого) и тест-кейс не завершен (в процессе проверки происходит ошибка).

Для описания тест-кейсов для проверки уровня базовых пользовательских требований может быть использован шаблон, представленный таблицей 12.

Таблица 12 – Тест-кейсы для проверки уровня базовых пользовательских требований

Идентификатор тест-кейса	**Заглавие тест- кейса**	**Шаги тест-кейса**	**Ожидаемый результат**
…	…	…	…

Общие требования к написанию шагов тест-кейса:

– заглавие тест-кейса должно быть информативным (понятно, что делает тест-кейс);

– при написании заглавия тест-кейса слует использовать безличную форму глагола;

– один тест-кейс должен проверять только одну функцию;

– шаги тест-кейса должны быть пронумерованы (даже если один шаг), описаны понятно и трактоваться одназначно;

– ожидаемый результат должен быть понятным;

– ожидаемые результаты обязательно следует приводить для каждого шага проверки.

В рамках лабораторной работы следует:

- разработать тест-кейсы для проверки уровня **всех базовых пользовательских требований** разработанного программного средства (перечень базовых пользовательских требований см. Задание 1 Лабораторная работа № 4[23]**)**;
- протестировать **все базовые пользовательские требования.**
- результаты тестирования оформить в виде таблицы 12.

[23] Тонкович, И.Н., **Разработка программного обеспечения: планирование, анализ и моделирование**: метод. пособие по дисциплине «Технологии проектирования сложных информационных систем» для студентов очной формы обучения учреждений высшего образования направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте) / И. Н. Тонкович, А. В. Шелест. – Репозиторий БГУИР, 2023. – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/50325.

Краткие теоретические сведения и методические указания к Заданию 2

Модульное тестирование (Unit Testing, unit-тестирование) – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Цель модульного тестирования – проверить, что каждая единица программного кода работает должным образом.

Данный вид тестирования выполняют на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

Отсутствие модульного тестирования при написании кода значительно увеличивает уровень дефектов при дальнейшем (интеграционном, системном, и приемочном) тестировании.

Модульное тестирование делят на **ручное** и **автоматизированное**. На практике чаще используется автоматизированное тестирование.

Алгоритм автоматизированного тестирования:

1 В приложение записывают единицу кода, чтобы протестировать ее. Делают комментарий и далее удаляют тестовый код при развертывании приложения.

2 Для проведения более качественного тестирования изолируют единицу кода, что подразумевает копирование кода в собственную среду тестирования. Данная процедура позволяет выявить ненужные зависимости между тестируемым кодом и другими компонентами (модулями) или пространствами данных в программном изделии.

3 Для разработки автоматизированных тестовых случаев применяют какой-либо UnitTest Framework. Используя инфраструктуру автоматизации, задаются критерии теста для проверки корректного выполнения кода, и в процессе выполнения тестовых случаев фиксируются неудачные. Большинство фреймворков автоматически выделяют и сообщают о неудачных тестах и могут остановить последующее тестирование, опираясь на серьезность сбоя.

4 Проводят модульное тестирование, алгоритм которого заключается в:

- создании тестовых случаев;
- просмотре / переработке;
- базовой линии;
- выполнении тестовых случаев.

```
// тестируемый класс
public class Calculator {

    public int getSum(int x, int y) {
        return x+y;
    }

    public int getDivide(int x, int y) {
        return x/y;
    }

    public int getMultiple(int x, int y) {
        return x*y;
    }

}

// тестирующий класс
public class CalculatorTest {

    private Calculator calculator;

    @BeforeClass
    public static void beforeClass() {
        System.out.println("Before CalculatorTest.class");
    }

    @AfterClass
    public  static void afterClass() {
        System.out.println("After CalculatorTest.class");
    }

    @Before
    public void initTest() {
        calculator = new Calculator();
    }

    @After
    public void afterTest() {
        calculator = null;
    }

    @Test
    public void testGetSum() throws Exception {
        assertEquals(15, calculator.getSum(7,8));
    }

    @Test
    public void testGetDivide() throws Exception {
        assertEquals(20, calculator.getDivide(100,5));
    }

    @Test
    public void testGetMultiple() throws Exception {

    }

    @Test(expected = ArithmeticException.class)
    public void divisionWithException() {
```

```
        calculator.getDivide(15,0);
    }

    @Ignore("Message for ignored test")
    @Test
    public void ignoredTest() {
        System.out.println("will not print it");
    }

    @Test(timeout = 500)
    public void timeStampTest() {
        while (true);
    }
}
```

В рамках лабораторной работы следует:

- разработать UNIT-тесты для проверки работоспособности **всех функциональных возможностей (**перечень см. Задание 1, таблица 10, Лабораторная работа № 4[24]) разработанного программного средства;
- протестировать **все функциональные требования.**
- результаты тестирования свести в таблицу 13, которая должна содержать следующие графы:
  - ✓ **Тестируемый тип** – функция, модуль, системы и т.д. (программная единица, подвергаемая тестированию);
  - ✓ **Дата проведения теста**;
  - ✓ **Результаты тестирования** – краткая информация о проведенном тесте и его результаты;
  - ✓ **Примечания** – дополнительная информация, полученная в процессе проведения тестирования программной единицы
- **оценить степень покрытия кода тестами** (см. источник [13]).

Таблица 13 – Результаты автоматизированного тестирования функциональности ПС

Тестируемый тип	Дата проведения теста	Результаты тестирования	Примечания
…	…	…	…

[24] Тонкович, И.Н., **Разработка программного обеспечения: планирование, анализ и моделирование**: метод. пособие по дисциплине «Технологии проектирования сложных информационных систем» для студентов очной формы обучения учреждений высшего образования направления специальности 1-40 05 01-10 Информационные системы и технологии (в бизнес-менеджменте) / И. Н. Тонкович, А. В. Шелест. – Репозиторий БГУИР, 2023. – [Электронный ресурс]. – Режим доступа: https://libeldoc.bsuir.by/handle/123456789/50325

При выполнении лабораторной работы можно использовать сторонние библиотеки, фреймворки и прочие дополнительные элементы. Если были использованы сторонние компоненты, на них должна быть приведена ссылка в тексте отчета по лабораторной работе в следующем виде: <название источника>:<URL>.

Весь код Unit-тестов должен быть представлен в публичном репозитории.

Контрольные вопросы к лабораторной работе № 11

1 Что такое тестирование и как его выполняют?
2 Какие виды тестирования бывают?
3 Что такое тест-кейс?
4 Каковы основные атрибуты тест-кейса?
5 Каким условиям должно удовлетворять качественное заглавие тес-кейса?
6 Какие категории ошибок выявляет тестирование на основе тест-кейсов?
7 Что представляют собой наборы тест-кейсов?
8 Каковы задачи модульного тестирования?
9 Какие принципы лежат в основе модульного тестирования?
10 Когда целесообразно использовать модульное тестирование?
11 Каковы особенности модульного тестирования?
12 Какие категории ошибок выявляет модульное тестирование?
13 Какие преимущества и недостатки имеет модульное тестирование?
14 Какие существуют типы тестов по покрытию?
15 Как измеряется покрытие кода?

Лабораторная работа № 12

Внедрение программного средства.
Оценка качества разработанного программного средства

Цели выполнения лабораторной работы:

- составить эксплуатационные документы программного средства
- оценить качество разработанного программного средства

Задание	Этапы выполнения задания
1 Разработать руководство по установке (развертыванию) программного средства	1 Составить руководство по установке (развертыванию)
2 Разработать руководство пользователя программного средства	1 Составить руководство пользователя программного средства
3 Оценить качество разработанного программного средства	1 Изучить серию международных стандартов SQuaRE 2 Представить количественную оценку характеристик качества программного средства
4 Сформировать отчет по лабораторной работе	Содержание отчета: 1 Титульный лист (приложение А) 2 Руководство по установке (развертыванию) программного средства 3 Руководство пользователя программного средства 4 Таблица результатов количественной оценки характеристик качества программного средства 5 Выводы

Краткие теоретические сведения и методические указания к Заданию 1

Руководство по установке (развертыванию) программного средства в большинстве случаев предназначено для сотрудников технического отдела организации-заказчика. В зависимости от масштаба разработанного программного средства, а также масштаба самой организации руководство по установке (развертыванию) может иметь различное содержание.

В рамках выполнения лабораторной работы необходимо представить руководство по установке (развертыванию) в соответствии со следующей

структурой[25]:

1 **Необходимые программы и компоненты.** Описать все программные и аппаратные компоненты, которые должны быть установлены на ПК конечного пользователя для корректной работы ПС.

Пример описания компонентов

Для успешной установки и запуска программного средства необходимо наличие следующих компонентов:

– операционная система семейства Linux;
– сервер базы данных Postgre SQL;
– веб-сервер ApacheTomcat;
– Java Development Kit.

2 **Последовательность установки.** Представить цепочку действий, выполнение которых приведет к установке разработанного ПС на ПК конечного пользователя.

Пример описания последовательности шагов установки

Для установки разработанного программного средства необходимо выполнить следующие шаги:

1 Назначить права для пользователя, под которым будет проводиться установка и настройка.
2 Установить и настроить Java.
3 Установить и настроить сервер приложений Apache Tomcat.
4 Установить и настроить сервер баз данных Postgre SQL.
5 Установить и настроить разработанное ПС.

3 **Состав дистрибутива.** Указать, какие элементы входят в дистрибутив для установки и развертывания ПС.

Пример описания состава дистрибутива

В поставляемый конечному пользователю дистрибутив входят следующие элементы:

– скрипт генерации пустой БД;
– файл ПС с расширением .exe.

[25] При формировании содержания руководства по установке (развертыванию) были использованы материалы ресурса https://rykovodstvo.ru/exspl/143299/index.html

4 **Распаковка дистрибутива.** Распаковка дистрибутива имеет место быть **только в случае**, если установка ПС осуществляется в определенные системные директории ОС. Если распаковка дистрибутива осуществляется в новую или создаваемую при установке дистрибутива директорию, то данный раздел руководства можно опустить.

Пример описания распаковки дистрибутива

Для распаковки дистрибутива необходимо выполнить следующие шаги:
- удалить содержимое папки /var/apache-tomcat-7.0.67/webapps/ROOT;
- распаковать имеющийся дистрибутив подсистемы в папку /var/apache-tomcat-7.0.67/webapps/ROOT.

5 **Восстановление БД из резервной копии.** В данном разделе необходимо представить процесс восстановления БД, необходимой для корректной работы ПС, из скрипта БД, поставляемого в составе дистрибутива ПС.

Пример описания процесса восстановления БД

Для восстановления БД ПС из резервной копии необходимо выполнить следующие шаги:
a) Для входа в консоль postgresql необходимо выполнить команду:

```
sudo -u postgres psql
```

Для создания пустой БД необходимо выполнить команду:

```
CREATE DATABASE tmp_crsmev_kf WITH ENCODING='UTF8' CONNECTION LIMIT=-
1;
```

b) В случае удачного выполнения вышеуказанной команды в консоли появится сообщение – CREATE DATABASE.
c) Для задания прав пользователю tomcat на свежесозданную БД необходимо выполнить команду:

```
GRANT ALL privileges ON DATABASE tmp_crsmev_kf TO tomcat;
```

d) В случае удачного выполнения вышеуказанной команды в консоли появится сообщение – GRANT.
e) Для выхода из консоли postgresql необходимо набрать \q и нажать клавишу Enter.
f) Для выполнения скрипта из файла db_new_1.txt необходимо выполнить команду (предполагается, что файл скрипта находится в папке /home/user):

```
sudo psql –U postgres tmp_crsmev_kf < /home/user/db_new_1.txt
```

g) Для восстановления БД из резервной копии crsmev_kf.backup необходимо выполнить команду (предполагается, что файл резервной копии находится в папке /home/user):

```
sudo pg_restore -U postgres -v -d tmp_crsmev_kf
/home/user/crsmev_kf.backup
```

h) Для выполнения скрипта из файла db_new_2.txt необходимо выполнить команду (предполагается, что файл скрипта находится в папке /home/user):

```
sudo psql -U postgres tmp_crsmev_kf < /home/user/db_new_2.txt
```

i) Для выхода из консоли необходимо набрать \q и нажать клавишу Enter.

6 **Настройка дистрибутива.** В данном разделе требуется представить описание всех действий, которые необходимо выполнить в случае, если требуется изменение каких-либо системных файлов или файлов конфигурации (например, необходимо изменить настройки файла конфигурации после восстановления БД из резервной копии).

Пример описания настроек дистрибутива

Для корректной работы ПС необходимо изменить настройки конфигурационных файлов следующим образом:

– в файле /var/apache-tomcat-7.0.67/conf/server.xml отредактировать группу параметров host следующим образом:

```
unpackWARs="true" autoDeploy="true"
xmlValidation="false" xmlNamespaceAware="false"
```

– в файле /var/apache-tomcat-7.0.67/webapps/ROOT/WEB-INF/sx-config.xml изменить параметры подключения к серверу баз данных путем редактирования секции database следующим образом:

```
driver="org.postgresql.Driver"
url="jdbc:postgresql://127.0.0.1/tmp_crsmev_kf?socketTimeout=60&
loginTimeout=60&connectTimeout=60&tcpKeepAlive=true"
username="tomcat"
password="12345678"
maxActive="100"
testWhileIdle="true"
testOnBorrow="false"
validationQuery="select 1"
minEvictableIdleTimeMillis="600000"
timeBetweenEvictionRunsMillis="20000"
maxWait="21000"
defaultTransactionIsolation="1"
stackTrace="true"
caseInsensitive="true"
```

7 **Проверка работоспособности ПС.** В данном разделе необходимо описать последовательность действий, выполнение которых будет сигнализировать об успешной установке ПС на ПК конечного пользователя.

Пример описания проверки работоспособности ПС

Для проверки работоспособности ПО необходимо:
– убедиться в том, что сервис postgresql запущен;
– убедиться в том, что сервер приложений tomcat запущен;
– используя любой интернет браузер перейти на адрес http://localhost:8080, где 8080 порт, указанный при установке Tomcat.
В случае успеха в окне браузера откроется домашняя страница ПС.

Краткие теоретические сведения и методические указания к Заданию 2

Руководство пользователя ПС – документ, основным назначением которого является предоставление конечным пользователям ПС информации о работе программного обеспечения.

В лабораторной работе необходимо представить описание всех операций обработки данных, которые может осуществлять разработанное ПС.

В начале руководства[26] необходимо представить таблицу с описанием функций и задач, которые может выполнять разработанное ПС. Пример описания функций и задач, выполняемых ПС, представлен в таблице 14.

Таблица 14 – Пример таблицы с описанием функций и задач

Функции	Задачи	Описание
Обеспечивает многомерный анализ в табличной и графической формах	Визуализация отчетности	В ходе выполнения данной задачи пользователю системы предоставляется возможность работы с выбранным отчетом из состава преднастроенных
	Формирование табличных и графических форм отчетности	В ходе выполнения данной задачи пользователю системы предоставляется возможность формирования собственного отчета в табличном или графическом виде на базе преднастроенных компонентов

Выполнение каждой задачи очень часто разделяется между различными операциями обработки данных. После составления таблицы функций, предоставляемых ПС, необходимо описать каждую операцию по обработке данных в соответствии со следующей структурой:
– наименование операции;
– условия, при соблюдении которых возможно выполнение операции;
– подготовительные действия;
– основные действия в требуемой последовательности;
– заключительные действия (если в них есть необходимость);

[26] При формировании содержания руководства по пользователя были использованы материалы ресурса https://www.prj-exp.ru/patterns/pattern_user_guide.php

– ресурсы, расходуемые на операцию.

Также можно снабжать описание операций иллюстрациями работы ПС, текстом сообщений в журналы логов и прочей дополнительной информацией. Пример описания операции представлен в таблице 15.

Таблица 15 – Пример описания операций, реализуемых ПС

Параметр	Описание
Задача «Визуализация отчетности»	
Операция 1	**Регистрация на портале ИАС КХД**
Условия, при соблюдении которых возможно выполнение операции	Компьютер пользователя подключен к корпоративной сети. Портал ИАС КХД доступен. ИАС КХД функционирует в штатном режиме
Подготовительные действия	На ПК пользователя необходимо выполнить дополнительные настройки, приведенные в п. 3.2 настоящего документа
Основные действия в требуемой последовательности	На иконке «ИАС КХД» рабочего стола произвести двойной щелчок левой кнопкой мышки. В открывшемся окне в поле «Логин» ввести имя пользователя, в поле «Пароль» ввести пароль пользователя. Нажать кнопку «Далее»
Заключительные действия	Не требуются.
Ресурсы, расходуемые на операцию	15-30 секунд.
Операция 2	**Выбор отчета**
Условия, при соблюдении которых возможно выполнение операции	Успешная регистрация на Портале ИАС КХД
Подготовительные действия	Не требуются
Основные действия в требуемой последовательности	1) в появившемся окне «Мастер создания рабочих книг» поставить точку напротив пункта «открыть существующую рабочую книгу» (рисунок 28); 2) выбрать нужную рабочую книгу и нажать кнопку «Откр.» (рисунок 29)
Заключительные действия	После завершения работы с отчетом необходимо выбрать пункт меню «Файл», далее выбрать пункт «Закрыть»
Ресурсы, расходуемые на операцию	15 секунд
Задача «Формирование табличных и графических форм отчетности»	
…	**…**

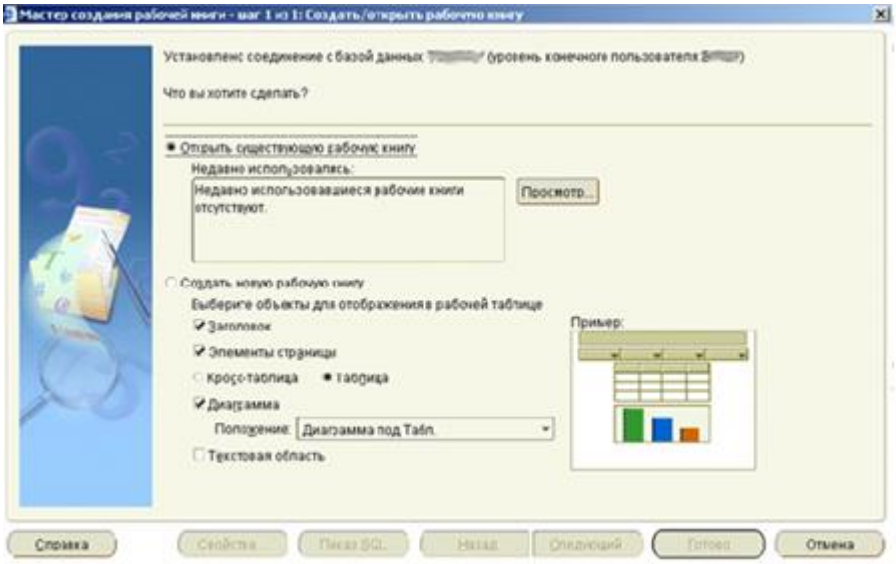


Рисунок 28 – Окно «Мастер создания рабочих книг»

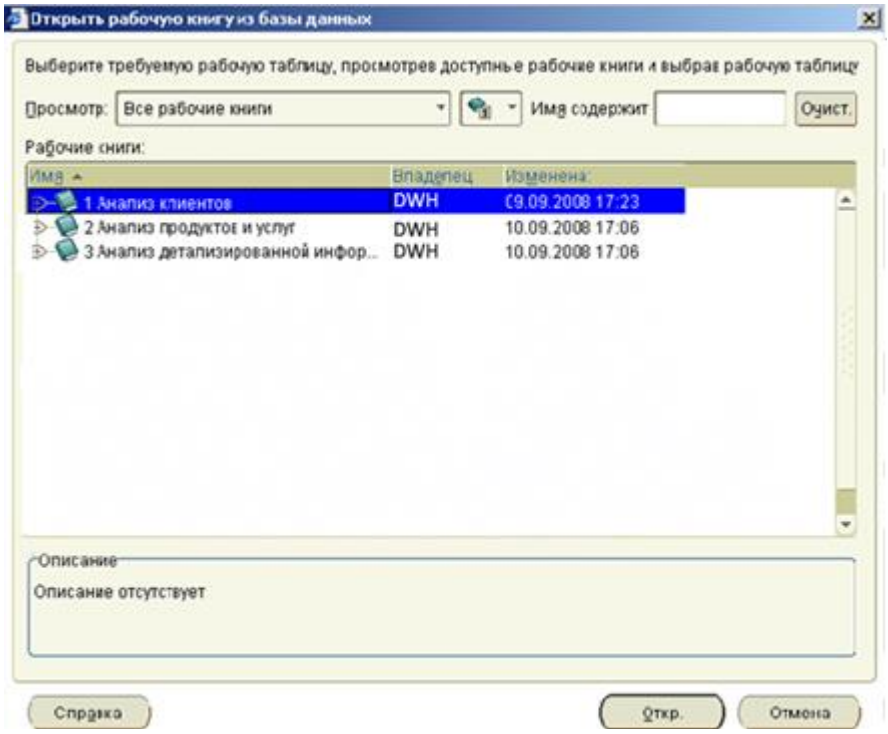


Рисунок 29 – Окно «Открыть рабочую книгу из базы данных»

Краткие теоретические сведения и методические указания к Заданию 3

Качество программного обеспечения (Software Quality) – это степень, в которой программное обеспечение обладает требуемой комбинацией свойств [1061-1998 IEEE Standard for Software Quality Metrics Methodology].

Качество системы – это степень удовлетворения системой **заявленных и подразумеваемых потребностей** различных заинтересованных сторон [ISO/IEC 25010:2011(ГОСТ Р ИСО/МЭК 25010-2015) Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программного обеспечения].

Качество реализации ПС представляет особую важность ввиду зависимости от него множества показателей бизнеса. На данный момент существует несколько стандартов, позволяющих специфицировать требования к качеству реализации ПС, а также описать и подготовить сами процедуры оценки качества.

Наиболее часто используемыми на практике являются стандарты: ISO/IEC 9126, ISO/IEC 14598, ГОСТ 28195-99, ГОСТ 28806-90, ИСО/МЭК 9126-2003, серия стандартов ISO/IEC 25000 (SQuaRE).

Указанные стандарты оперируют следующими терминами в области оценки качества [14, с. 147-148]:

1 **Атрибут** (attribute) – внутренне присущее свойство объекта, которое может быть распознано количественно или качественно человеком или автоматизированными средствами. Атрибуты могут быть внешними или внутренними.

2 **Качество** (quality) – совокупность характеристик программного продукта, относящаяся к его способности удовлетворять установленные и подразумеваемые потребности.

3 **Отказ** (failure) – прекращение способности продукта выполнять требуемую функцию или его неспособность работать в пределах заданных ограничений.

4 **Оценка качества** (quality evaluation) – систематическое исследование степени, в которой продукт способен к выполнению указанных требований.

5 **Ошибка** (fault) – некорректный шаг, процесс или определение данных в программе.

6 **Подразумеваемые потребности** (implied needs) – потребности, которые не были заданы, но являются реально существующими потребностями при использовании продукта в конкретных условиях. Подразумеваемые потребности включают потребности не заданные, но подразумеваемые другими

заданными потребностями, и те незаданные потребности, которые рассматриваются как очевидные. Некоторые потребности становятся очевидными при использовании продукта в конкретных условиях.

7 **Подхарактеристика качества** программного средства (software quality subcharacteristic) – характеристика качества программного средства, входящая в состав другой характеристики качества.

8 **Характеристика качества** программного средства (software quality characteristic) – категория свойств (атрибутов) программного средства, с помощью которых описывается и оценивается его качество. Характеристики качества программных средств могут быть определены с помощью подхарактеристик и в конечном итоге атрибутов качества программного средства.

9 **Шкала** (scale) – набор значений с определенными свойствами.

При оценке качества используются следующие типы шкал[27]:

– **номинальная** – соответствует набору категорий; классифицирует программы по признаку наличия или отсутствия некоторого свойства без учета градаций (например «да», «нет»);

– **порядковая** (упорядоченная) – соответствует упорядоченному набору делений шкалы; позволяет ранжировать свойства путем сравнения с опорными значениями; имеет небольшое количество делений (например, шкала с четырьмя градациями «отлично», «хорошо», «удовлетворительно», «неудовлетворительно», с двумя градациями, «удовлетворительно», «неудовлетворительно»);

– **интервальная** – соответствует упорядоченной шкале с равноудаленными делениями; обычно содержит достаточно большое количество делений с количественными значениями (например шкала с делениями 0, 1, 2, …, 10);

– **отношений** – соответствует упорядоченной шкале с равноудаленными делениями, содержащей значение нуля, представляющего полное отсутствие атрибута.

Два первых типа шкал применяются для оценки качественных атрибутов ПС, которые нельзя измерить количественно, и для ранжирования измеренных значений, третий и четвертый типы – для оценки количественных атрибутов [14, с. 148].

Вышеописанные стандарты регламентируют оценку качества ПС на основании того или иного вида модели качества.

В настоящий момент показатель качества регулируется международным стандартом ISO/IEC 25010:2011.

Заявленные и подразумеваемые потребности представлены в международных стандартах серии SQuaRE посредством моделей качества:

– модель качества при использовании (рисунок 30);

– модель качества продукта (рисунок 31).

[27] Более детальная информация о типах шкал представлена в [23, с. 231–233].

Рисунок 30 – Модель качества при использовании

Рисунок 31 – Модель качества при использовании

Ввиду сложности, возникающей в ходе оценки качественных характеристик ПС по причине их субъективности, в рамках лабораторной работы необходимо провести оценку качественных характеристик ПС с использование иерархической модели оценки качества ПС (рисунок 32).

Значения той или иной подхарактеристики целесообразно представлять в относительных единицах, для вычисления которых следует использовать одну из ниже представленных формул [14, с.267]:

$$X = \frac{A}{B} \quad (1)$$

или

$$X = 1 - \frac{A}{B} \quad (2)$$

где X – значение метрики; A – абсолютное (измеренное) значение некоторого свойства (атрибута) оцениваемого продукта, системы или документации; B – базовое значение соответствующего свойства

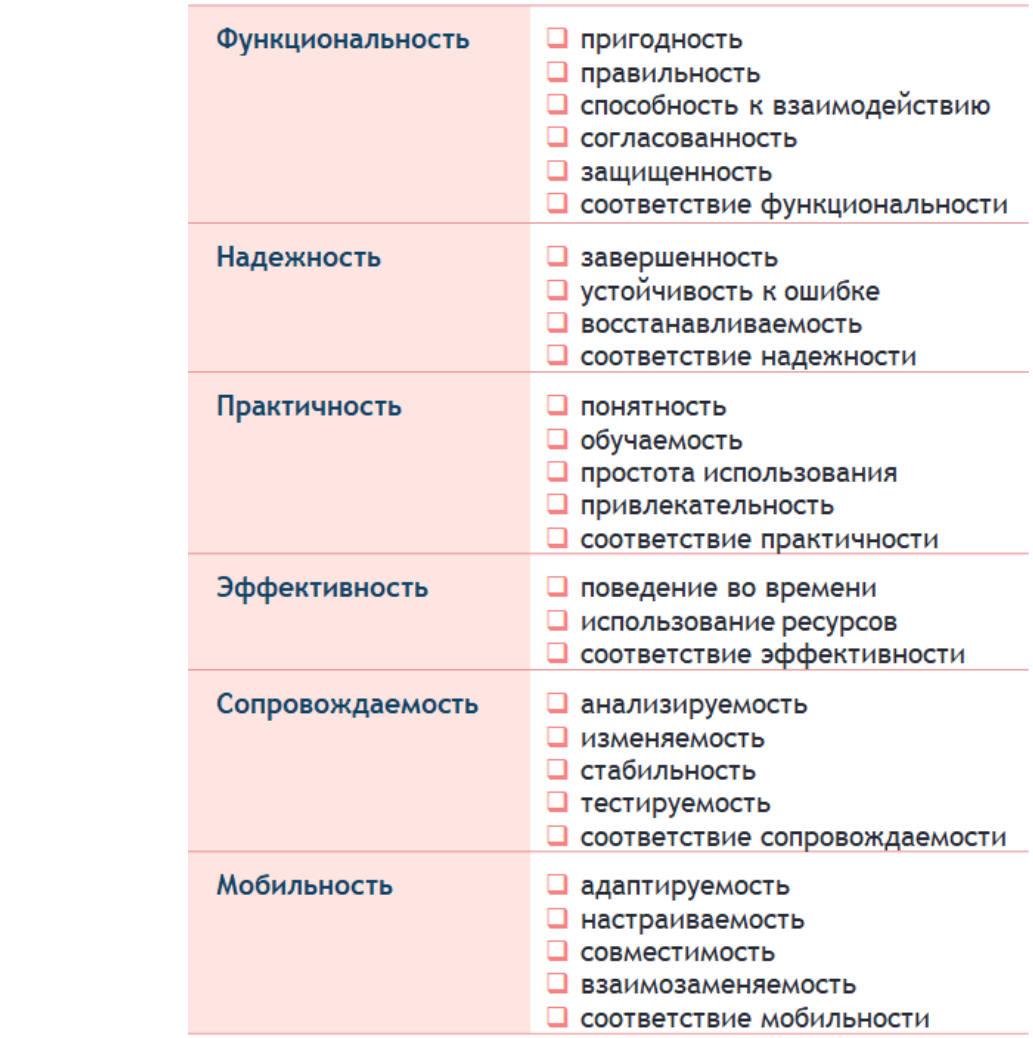


Рисунок 32 – Иерархическая модель оценки качества ПС

Результаты вычисления значений подхарактеристик следует свести в таблицу 16.

Таблица 16 – Результаты количественной оценки характеристик качества ПС[28]

Название подхарактеристики	**Относительное значение подхарактеристики**
Функциональность	
Пригодность	0,74
…	…
Надежность	
Завершенность	0,52
…	…

[28] См. пример в п.14.3 [14]

Контрольные вопросы к лабораторной работе № 12

1 В каком стандарте устанавливаются виды программ и программных документов для вычислительных машин, комплексов и систем независимо от их назначения и области применения?

2 Какие существуют виды программных документов?

3 Для чего предназначены эксплуатационные документы?

4 Какие виды эксплуатационных документов выделяют?

5 Какие элементы включает в себя руководство по установке (развертыванию) программных разработок?

6 Что такое руководство пользователя и для чего его создают?

7 Какова структура руководства пользователя?

8 В чем особенности разработки и структуры руководства пользователя для программного обеспечения?

9 Какие популярные инструменты используются для создания качественного руководства?

10 Что такое качество систем и программного обеспечения?

11 Зачем нужны модели качества?

12 Какими тремя главными аспектами характеризуется качество программного обеспечения?

13 Что представляет собой модель качества при использовании?

14 Что представляет собой модель качества продукта?

15 Какова связь модели качества при использовании и модели качества продукта?

16 Какие внутренние и внешние характеристика качества важны для программистов?

17 Как характеристики качества программного обеспечения влияют на процесс проектирования?

Библиографический список

Список использованных источников

1 Материал по разработке пользовательского интерфейса [Электронный ресурс]. – Режим доступа : https://livetyping.com/ru/blog/chto-takoe-razrabotka-polzovatelskogo-interfeisa-i- zachem-tt-zakazyvat.

2 Принципы разработки пользовательского интерфейса [Электронный ресурс]. – Режим доступа : https://clck.ru/DcKAK.

3 Этапы разработки пользовательского интерфейса [Электронный ресурс]. – Режим доступа : https://vc.ru/design/58502- etapy-razrabotki-polzovatels-kogo-interfeysa-kak-sdelat-tak-chtoby-ui-ne-lishil-vas-pribyli.

4 Дейт, К. Дж. Введение в системы баз данных, 8-е издание / К. Дж. Дейт. – М.: Издательский дом «Вильяме», 2005. – 1328 с.

5 Гамма, Э. Паттерны объектно-ориентированного проектирования / Э. Гамма [и др.]. – СПб.: Питер, 2020 – 448 с.

6 Интернет-ресурс о паттернах проектирования [Электронный ресурс]. – Режим доступа : https://refactoring.guru/ru.

7 Буч Г. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2006. – 496 с.

8 ГОСТ 19.701-90 [Электронный ресурс]. – Режим доступа : https://www.swrit.ru/doc/espd/19.701-90.pdf/.

9 ГОСТ 19.002-80 [Электронный ресурс]. – Режим доступа : https://www.swrit.ru/doc/espd/19.002-80.pdf/.

10 Куликов, С. С, Тестирование программного обеспечения: учеб. пособие / С. С. Куликов [и др.]. – Минск : БГУИР, 2019. – 276 с. : ил.

11 SOLID Principles: Explanation and examples [Электронный ресурс]. – Режим доступа : https://itnext.io/solid-principles-explanation-and-examples-715b975dcad4

12 Design and Build Great Web APIs. Robust, Reliable, and Resilient / Raleigh. – North Ca-rolina, 2020.

13 Покрытие кода [Электронный ресурс]. – Режим доступа : https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/pokrytie-koda-2.

14 Бахтизин, В. В. Метрология, стандартизация и сертификация в информационных технологиях : учеб. пособие. В 2 ч. Ч. 2 / В. В. Бахтизин, Л. А. Глухова. – Минск : БГУИР, 2016. – С. 141–343.

Дополнительная литература

1 Code Coverage Tutorial: Branch, Statement, Decision, FSM [Электронный ресурс]. – Режим доступа : https://www.guru99.com/code-coverage.html.

2 Getting Started With Testing in Python [Электронный ресурс]. – Режим доступа : https://realpython.com/python-testing/.

3 How to become a better programmer [Электронный ресурс]. – Режим доступа : https://medium.com/@derodu/design-patterns-kiss-dry-tda-yagni-soc-828c112b89ee.

4 REST API Best Practices [Электронный ресурс]. – Режим доступа : https://habr.com/ru/post/351890/.

5 REST API Tutorial [Электронный ресурс]. – Режим доступа : https://www.restapitutorial.com/.

6 REST: простым языком [Электронный ресурс]. – Режим доступа : https://cutt.ly/llljEEh.

7 SOLID principles [Электронный ресурс]. – Режим доступа : https://siderite.dev/blog/solid-principles-plus-dry-yagni-kiss.html/.

8 Unit-тесты на C# [Электронный ресурс]. – Режим доступа : https://vc.ru/dev/137335-unit-testy-na-c.

9 Автоматические тесты при помощи chai и mocha [Электронный ресурс]. – Режим доступа : https://learn.javascript.ru/testing.

10 Все о Unit testing: методики, понятия, практика [Электронный ресурс]. – Режим доступа : https://javarush.ru/groups/posts/2500-vse-o-unit-testing-metodiki-ponjatija-praktika.

11 Макконнелл, С. Совершенный код. Мастер-класс / С. Макконнелл. – СПб.: БХВ, 2022. – 896 с.

12 Мартин, Р. Чистый код. Создание, анализ и рефакторинг / Р. Мартин. – СПб.: Питер, 2020. – 464 с.

13 Подходы к проектированию RESTful API [Электронный ресурс]. – Режим доступа : https://dataart.com.ua/news/podhody-k-proektirovaniyu-restful-api/.

14 Проектирование API REST [Электронный ресурс]. – Режим доступа : https://www.ibm.com/support/knowledgecenter/ru/SS4SVW_3.0.0/designing/designingapis.html.

15 Рекомендации по REST API – примеры проектирования веб-сервисов на Java и Spring [Электронный ресурс]. – Режим доступа : https://habr.com/ru/post/483374/.

16 Руководство по PHPUnit [Электронный ресурс]. – Режим доступа : https://phpunit.readthedocs.io/ru/latest/.

17 Самые важные архитектурные шаблоны [Электронный ресурс]. – Режим доступа : https://habr.com/ru/company/alconost/blog/522662/.

18 Юнит-тесты [Электронный ресурс]. – Режим доступа : https://habr.com/ru/company/tensor/blog/347358/.

ПРИЛОЖЕНИЕ А
(обязательное)
Пример титульного листа отчета
по лабораторной работе

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования
Кафедра проектирования информационно-компьютерных систем

Отчет
по лабораторной работе №X
на тему:
НАЗВАНИЕ ЛАБОРАТОРНОЙ РАБОТЫ

Проверил _____ И.И. Иванов
(подпись)

Выполнил _____ С.С. Сидоров
(подпись) номер группы

Минск, 202X

Электронный ресурс

Учебное издание

Тонкович Ирина Николаевна
Шелест Анна Вадимовна

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ: ПРОЕКТИРОВАНИЕ, КОНСТРУИРОВАНИЕ И ВНЕДРЕНИЕ

Методическое пособие
по дисциплине «Технологии проектирования
сложных информационных систем»
для студентов очной формы обучения учреждений высшего
образования направления специальности
1-40 05 01-10 Информационные системы и технологии
(в бизнес-менеджменте)

Ответственный за выпуск *А.В. Шелест*
Компьютерная верстка, правка, оригинал-макет *А.В. Шелест*