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

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
старший преподаватель
_____ Т.М. Унучек
___.____.2021

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему:

«ПРОЕКТНЫЙ МЕНЕДЖМЕНТ И ПРОГРАММНАЯ ПОДДЕРЖКА УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА»

БГУИР КП 1-40 01 02-08 041 ПЗ

Выполнил студент группы 914301
ПЕТУШОК Андрей Иванович

_____
(подпись студента)

Курсовой проект представлен на проверку 01.12.2021

_____
(подпись студента)

Минск 2021

РЕФЕРАТ

БГУИР КП 1-40 05 01 041 ПЗ

Петушок, А.И. Проектный менеджмент и программная поддержка управления сроками и стоимостью проекта / А.И. Петушок. – Минск: БГУИР, 2021. – 54 с.

Пояснительная записка 54 с., 22 рис., 9 источников, 5 приложений

ДОБАВЛЕНИЕ, ПРОГРАММНЫЙ МЕНЕДЖМЕНТ, УПРАВЛЕНИЕ, РЕДАКТИРОВАНИЕ ДАННЫХ, ПОЛЬЗОВАТЕЛЬ, АДМИНИСТРАТОР, ФИРМА, ОРГАНИЗАЦИЯ, ДОБАВЛЕНИЕ, УДАЛЕНИЕ, БАЗА ДАННЫХ, СРОКИ, СТОИМОСТЬ

Цель проектирования: проектирование графического пользовательского интерфейса для управления проектами.

Методология проведения работы: в процессе решения поставленных задач использованы принципы системного подхода, теория разработки JavaFX, аналитические методы, методы компьютерной обработки экспериментальных данных, методы работы с базой данных.

Результаты работы: выполнен анализ литературно-патентных исследований, рассмотрено общетехническое обоснование разработки электронного модуля; проведён анализ спроектированной программы; осуществлено моделирование физических процессов, протекающих в процессе работы программы, разработана графическая часть проекта. Модуль предназначен для управления проектами фирмы.

Структуру управления администратор определяет и задает самостоятельно на стадии создания компании и работников для базы данных, пользователь же использует эту структуру для просмотра проектов. Структура предполагает формы с определёнными возможностями для каждого пользователя. Она включает в себя редактирование данных о проекте, пользователях, авторизацию, анализ и обработку данных.

Область применения результатов: могут быть использованы небольшими фирмами, занимающиеся разработкой программных продуктов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире очень быстро развивается отрасль информационных технологий. Наша жизнь становится заметно проще благодаря программам и программным обеспечениям, которые в некоторой степени автоматизируют жизненные процессы. Банки, заказ еды, дизайн, медицина и многие другие отрасли постепенно переходят на информационный уровень организации своих процессов.

Во многих организациях системы управления проектами, наряду с традиционными офисными технологиями, сегодня являются элементом стандартной конфигурации рабочего места каждого сотрудника.

Для создания таких программных продуктов необходимо некоторое количество людей с определённым знаниями в ИТ сфере. Даже самые простые проекты не обходятся без структуризации процесса производства продукта. Кроме того, необходимо руководить этим процессом. И здесь тоже не обошлось бес автоматизации. Создаются программы, которые облегчают жизнь программистам, помогают разделить работу и скооперироваться, когда это необходимо.

Необходимо, чтобы такие программные обеспечения позволяли хранить и передавать необходимые объёмы информации с высокой скоростью. Также должен быть простой и понятный дизайн, чтобы работать было приятно и не было необходимости долго разбираться в функционале. Нужно, чтобы в ПО можно было заниматься менеджментом разрабатываемого проекта. А именно руководить сроками, стоимостью и работой команды разработчиков.

Целью данного курсового проекта является оптимизация работы над программным проектом, внесение в базу фирмы его стоимости и сроков выполнения.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

В этой работе проектируется база данных «WorkManage».

Проектный менеджмент – это интеграция информационных систем планирования с управленческими процедурами и организационной структурой предприятия. Программой, выполняющей данные условия в этой программной области, является WorkManage.

Проектный менеджмент означает реализацию определенных специальных задач внутри существующей структуры предприятия или между различными предприятиями, при которых, по возможности, не должно быть оказано отрицательное воздействие на исходные производственные задачи. Применение проектных методов является одним из элементов перехода организаций к современным структурам управления, самоуправляемым командам, саморегулирующимся организационным структурам и другим новым управленческим решениям.

Проекты нацелены на получение определенных результатов – иными словами, они направлены на достижение целей. Именно эти цели являются движущей силой проекта, и все усилия по его планированию и реализации предпринимаются для того, чтобы эти цели были достигнуты. Проект обычно предполагает целый комплекс взаимосвязанных целей. Например, основной целью проекта, связанного с компьютерным программным обеспечением, может быть разработка информационной системы управления предприятием. Промежуточными целями (подцелями) могут быть разработка базы данных, разработка математического и программного обеспечения, тестирование системы. В разработке базы данных, в свою очередь, также могут быть выделены цели более низкого уровня - разработка логической структуры базы данных, реализация базы данных с помощью СУБД, загрузка данных и так далее. Тот факт, что проекты ориентированы на достижение цели, имеет огромный внутренний смысл для управления ими. Прежде всего, он предполагает, что важной чертой управления проектами является точное определение и формулирование целей, начиная с высшего уровня, а затем постепенно опускаясь до наиболее детализированных целей и задач.

Проекты сложны уже по самой своей сути. Они включают в себя выполнение многочисленных взаимосвязанных действий. В отдельных случаях эти взаимосвязи достаточно очевидны (например, технологические зависимости), в других случаях они имеют более тонкую природу. Некоторые промежуточные задания не могут быть реализованы, пока не завершены

другие задания; некоторые задания могут осуществляться только параллельно, и так далее. Если нарушается синхронизация выполнения разных заданий, весь проект может быть поставлен под угрозу. Если немного задуматься над этой характеристикой проекта, становится очевидно, что проект – это система, то есть целое, складывающееся из взаимосвязанных частей, причем система динамическая, и, следовательно, требующая особых подходов к управлению.

Проекты выполняются в течение конечного периода времени. Они временны. У них есть более или менее четко выраженные начало и конец. Проект заканчивается, когда достигнуты его основные цели. Значительная часть усилий при работе с проектом направлена именно на обеспечение того, чтобы проект был завершен в намеченное время. Для этого готовятся графики, показывающие время начала и окончания заданий, входящих в проект. Отличие проекта от производственной системы заключается в том, что проект является однократной, не циклической деятельностью. Серийный же выпуск продукции не имеет заранее определенного конца во времени и зависит лишь от наличия и величины спроса. Когда исчезает спрос, производственный цикл кончается. Производственные циклы в чистом виде не являются проектами. Однако, в последнее время проектный подход все чаще применяется и к процессам, ориентированным на непрерывное производство. Например, проекты увеличения производства до указанного уровня в течении определенного периода, исходя из заданного бюджета, или выполнение определенных заказов, имеющих договорные сроки поставки. Проект как система деятельности существует ровно столько времени, сколько его требуется для получения конечного результата. Концепция проекта, однако, не противоречит концепции фирмы или предприятия и вполне совместима с ней. Напротив, проект часто становится основной формой деятельности фирмы.

Проекты - мероприятия неповторимые и однократные. Вместе с тем, степень уникальности может сильно отличаться от одного проекта к другому. Если вы занимаетесь строительством коттеджей и возводите двадцатый по счету однотипный коттедж, степень уникальности вашего проекта достаточно невелика. Базовые элементы этого дома идентичны элементам предыдущих девятнадцати, которые вы уже построили. Основные же источники уникальности, однако, могут быть заложены в специфике конкретной производственной ситуации - в расположении дома и окружающего ландшафта, в особенностях поставок материалов и комплектующих, в новых субподрядчиках. С другой стороны, если вы разрабатываете уникальный прибор или технологию, вы, безусловно, имеете дело с задачей весьма

уникальной. Вы делаете то, что никогда раньше не делалось. И поскольку прошлый опыт может в данном случае лишь ограниченно подсказывать вам, чего можно ожидать при выполнении проекта, он полон риска и неопределенности.

Под проектом понимается уникальный комплекс взаимосвязанных мероприятий, направленных на достижение конкретной цели при определенных требованиях к срокам, бюджету и характеристикам ожидаемых результатов. В некоторых источниках проект называют расписанием.

Заказчик, приходя в фирму по разработке программных продуктов, должен указать весь необходимый функционал своего будущего продукта, описать как будет выглядеть программа, сойтись с исполнителем в цене. Нередко некоторые работы проекта нужно привязать к реальной календарной дате. Например, представитель заказчика приезжает 15 сентября для ознакомления с разрабатываемой программой. Поэтому работа "Подготовка демонстрационной версии" должна быть закончена не позднее 15 сентября. Подобная привязка работы к дате называется ее ограничением. Вся эта информация должна быть где-то записана, обработана и представлена команде разработчиков в правильном и юзабельном для них формате. Также должна быть база сотрудников фирмы, которая будет заниматься выполнением заказа, чтобы можно было направить на определённый заказ необходимое количество специалистов.

В «WorkManage» ведется архив, хранящий информацию о всех выполненных заказах.

Цель проекта раскрывается и "проецируется" на поверхности подробнейшего и детального плана действий, который может включать в себя различные аспекты проекта и выражаться в зависимости от самого проекта в различных документах, таких как иерархическое "дерево целей", структура работ, структура стоимости, структура продукции (результата) проекта, сетевые и информационно-технологические модели. Кроме этого, тщательной проработке подвергаются средства и предметы деятельности, необходимые для реализации проекта: основные средства, ресурсы проекта, организационная структура проекта, система коммуникаций между элементами проекта и пр.

2 ПОСТАНОВКА ЗАДАЧИ ПРОЕКТИРОВАНИЯ И РАЗРАБОТКА СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

2.1 Детализация задач в области разработки системы

Для решения поставленной задачи необходимо разработать комфортную для использования IT-систему, при помощи которой можно будет вести учёт проектов, сотрудников, управлять сроками и стоимостью проектов.

В приложении необходимо реализовать:

– интуитивно понятный интерфейс для комфортного пользования приложением;

– авторизация пользователей (для разделения прав доступа администратору и пользователю);

– администратор имеет доступ к таблицам всех проектов, пользователей, работников;

– обработка исключений, которые могут возникнуть при использовании приложения.

2.2 Обзор методов решения поставленных задач

Задача курсового проекта решалась с помощью приложения, в основе работы которого лежит так называемая модель взаимодействия клиент-сервера на основе TCP соединения. Было реализовано GUI-приложение, которое работает в связке с сервером, а тот в свою очередь с базой данных. В качестве СУБД используется PostgreSQL.

Вся реализация производилась на языке Java, так как он является объектно-ориентированным и имеет в комплектации достаточно объемную библиотеку классов, которая значительно облегчает разработку программного обеспечения, предлагая программисту мощные средства решения распространенных задач. В проекте основная среда разработки — IntelliJ IDEA.

С помощью Draw.io были созданы UML-диаграммы. Выбор был остановлен на нём в связи с понятным интерфейсом: слева набор блоков, справа настройки внешнего вида и связей, в центре сам редактор.

В данной работе использовались два паттерна проектирования:

– singletone;
– delegate.

В курсовом проекте присутствует авторизация пользователей, возможность просмотра, редактирования, удаления данных и создания новых записей.

Для выполнения UML-моделей в стандарте IDEF0 использовалось CASE-средство CA AllFusion Process Modeler r7 (BPwin). Для информационного моделирования применялось средство CA AllFusion ERwin Data Modeler r7 (ERwin).

2.2.1 Паттерн проектирования «Singletone»

Само описание паттерна достаточно простое — класс должен гарантированно иметь лишь один объект, и к этому объекту должен быть предоставлен глобальный доступ. Скорее всего, причина его популярности как раз и кроется в этой простоте — всего лишь один класс, ничего сложного. Это, наверное, самый простой для изучения и реализации паттерн. Если вы встретите человека, который только что узнал о существовании паттернов проектирования, можете быть уверены, что он уже знает про Singleton. Проблема заключается в том, что, когда из инструментов у вас есть только молоток, всё вокруг выглядит как гвозди. Из-за этого «Одиночкой» часто злоупотребляют.

В объектно-ориентированном программировании существует правило хорошего тона — (Single Responsibility Principle, первая буква в аббревиатуре SOLID). Согласно этому правилу, каждый класс должен отвечать лишь за один какой-то аспект. Совершенно очевидно, что любой Singleton-класс отвечает сразу за две вещи: за то, что класс имеет лишь один объект, и за реализацию того, для чего этот класс вообще был создан.

Принцип единственной обязанности был создан не просто так — если класс отвечает за несколько действий, то, внося изменения в один аспект поведения класса, можно затронуть и другой, что может сильно усложнить разработку. Так же разработку усложняет тот факт, что переиспользование (reusability) класса практически невозможно. Поэтому хорошим шагом было бы, во-первых, вынести отслеживание того, является ли экземпляр класса единственным, из класса куда-либо во вне, а во-вторых, сделать так, чтобы у класса, в зависимости от контекста, появилась возможность перестать быть Singleton’ом, что позволило бы использовать его в разных ситуациях, в

зависимости от необходимости (т.е. с одним экземпляром, с неограниченным количество экземпляров, с ограниченным набором экземпляров и так далее).

2.2.2 Паттерн проектирования «Delegate»

Паттерн делегирования является поведенческим (behavioral) паттерном проектрования.

Это техника, в которой объект выражает определенное поведение снаружи, но в реальности делегирует ответственность за реализацию этого поведения связанному объекту.

Паттерн делегирования обеспечивает механизм отвлечения от реализации и контроля желаемого действия. Класс, вызываемый для выполнения действия, не выполняет его, а фактически делегирует вспомогательному классу. Потребитель не имеет или не требует знания фактического класса, выполняющего действие, только контейнера, к которому производится вызов.

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

Описание данной предметной области произведено посредством построения IDEF0-модели в системе Erwin Process Modeler.

Главный концептуальный принцип методологии IDEF - представление любой изучаемой системы в виде набора взаимодействующих и взаимосвязанных блоков, отображающих процессы, операции, действия, происходящие в изучаемой системе. В IDEF0 все, что происходит в системе и ее элементах — это функции, каждая из которых ставится в соответствие блок, который представляет собой прямоугольник.

Для IDEF0 имеет значение сторона процесса и связанная с ней стрелка:

– слева входящая стрелка – вход бизнес-процесса – информация (документ) или ТМЦ, который будет преобразован в ходе выполнения процесса;

– справа исходящая стрелка – выход бизнес-процесса – преобразованная информация (документ) или ТМЦ;

– сверху входящая стрелка – управление бизнес-процесса – информация или документ, который определяет, как должен выполняться бизнес-процесс, как должно происходить преобразование входа в выход;

– снизу входящая стрелка – механизм бизнес-процесса – то, что преобразовывает вход в выход: сотрудники или техника. Считается, что за один цикл процесса не происходит изменения механизма.

Методология описания бизнес-процессов IDEF0 наиболее широко используется, так как в ней рассматриваются логические отношения между работами.

Входными данными системы будет являться заказ на выполнение проекта.

Результатом системы – выходом – будет служить добавленный проект.

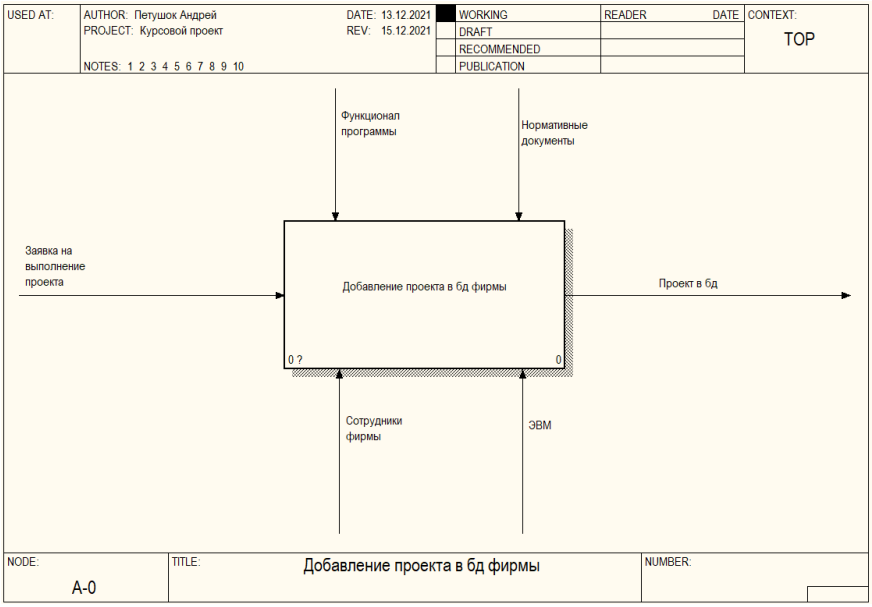

Рисунок 3.1 – Контекстная диаграмма добавления проекта

Для того чтобы описать работу функционального блока более подробно, выполняется его декомпозиция, и моделируется диаграмма второго уровня, которая представлена на рисунке 3.2. На ней представлено три функциональных блока декомпозиции.

Данный уровень включает в себя:

- анализ входных данных;
- внесение данных в бланк;
- добавление проекта в базу данных.

Конечная цель системы: добавленный проект в базу.

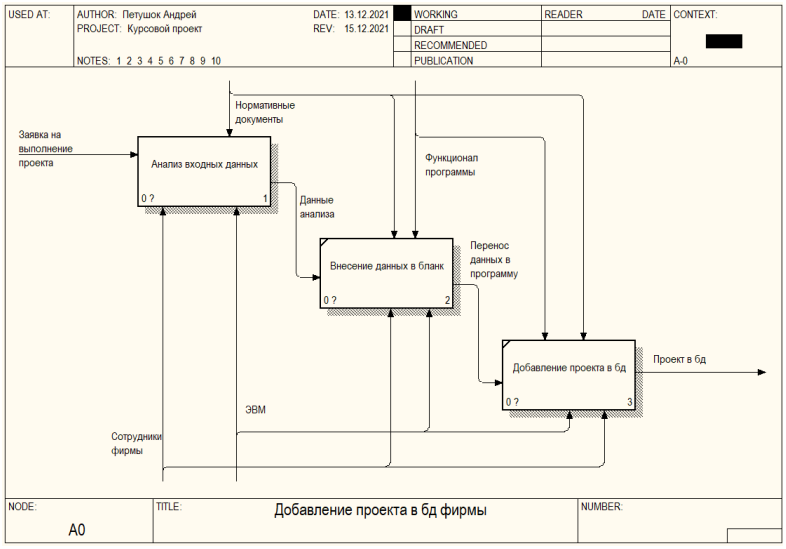

Рисунок 3.2 – Декомпозиция функционального блока

Далее следует декомпозиция процесса «Анализ входных данных», результат которой показан на рисунке 3.3.

На данном уровне выделены следующие подпроцессы:

- определение стоимости проекта;
- определение сроков написания технического задания;
- оценка времени на реализацию проекта.

Итоговая цель: данные о разработке, сроках и стоимости проекта.

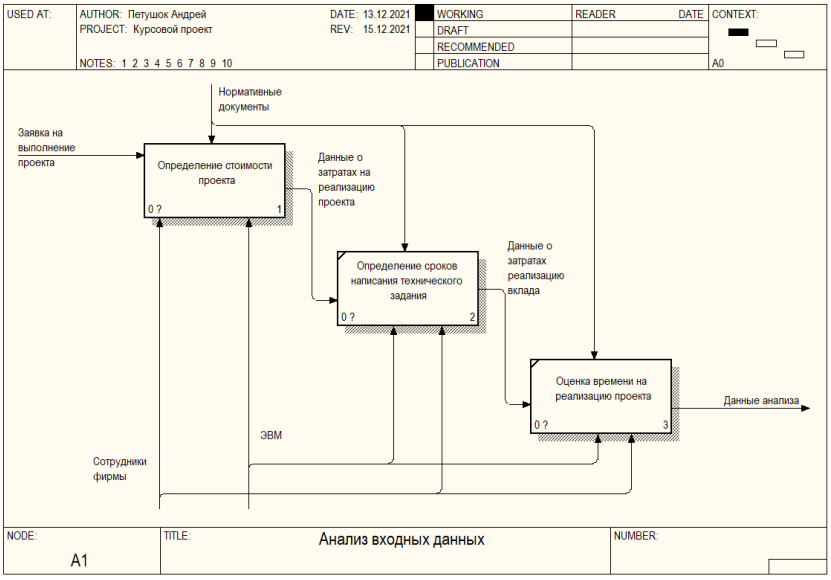

Рисунок 3.3 – Анализ входных данных

На следующем этапе разработки модели выполняется декомпозиция подпроцесса «Определение стоимости проекта», показанная на рисунке 3.4.

В данном уровне существуют следующие подпроцессы:

- формирование команды;
- оценка стоимости использования сторонних программных ресурсов;
- оценка объёмов работы над программным продуктом.

Итоговая цель подпроцесса: затраты на производство проекта.

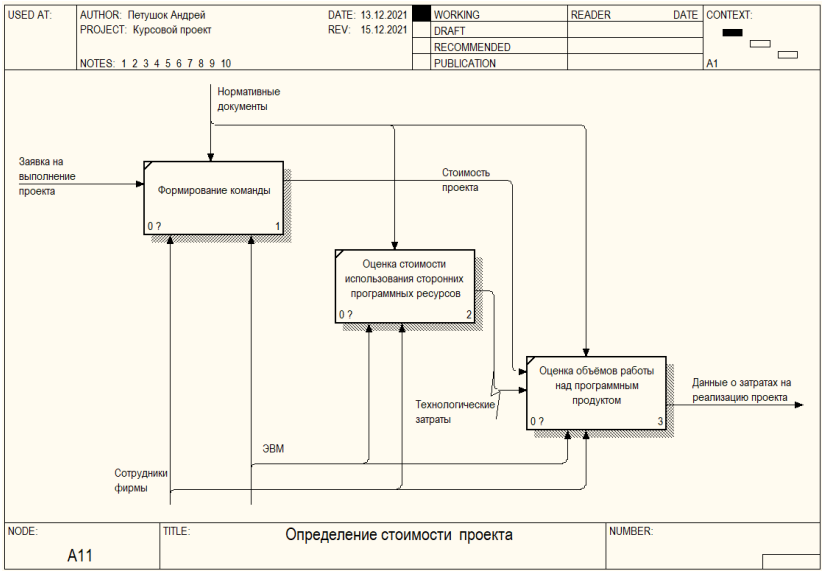

Рисунок 3.4 – Определение затрат на производство

В результате последовательного выполнения всех процессов происходит добавление итоговой информации о проекте в базу.

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

При проектировании системы проектного менеджмента было сформировано 5 сущностей, а именно:

– пользователь;
– проект;
– архив выполненных проектов;
– работник;
– присоединение к проекту.

Сущность Пользователь нужна для авторизации пользователей и содержит в себе следующие атрибуты:

– user_id – хранит уникальный номер пользователя;
– user_email – логин пользователя, необходимый для авторизации;
– user_name – хранит имя пользователя;
– user_lastName – хранит отчество пользователя;
– user_password – пароль пользователя, нужный для авторизации;
– user_role – роль пользователя.

Сущность Проект необходима для учета проектов:

– project_id – уникальный номер проекта;
– project _userId – атрибут унаследованный от сущности Пользователь;
– project _name – имя проекта;
– project _cost – стоимость проекта;
– project _about – информация о проекте;
– project _time – информация о сроках выполнения проекта.

Сущность Работник содержит такие атрибуты как:

– worker_id – атрибут унаследованный от сущности Пользователь;
– worker_position – должность работника.

Сущность Присоединение к проекту содержит такие атрибуты как:

– total_id – уникальный номер проекта;
– total_userId – атрибут унаследованный от сущности Пользователь;
– total_name – имя проекта;
– total_cost – стоимость проекта;
– total_about – информация о проекте;
– total_time – информация о сроках выполнения проекта.

Сущность Архив выполненных проектов содержит такие атрибуты как:

– done_id – уникальный номер проекта;
– done_name – имя проекта;
– done_cost – стоимость проекта;

Рассмотрим модель разработанную логическую модель системы (см. рисунок 4.1).

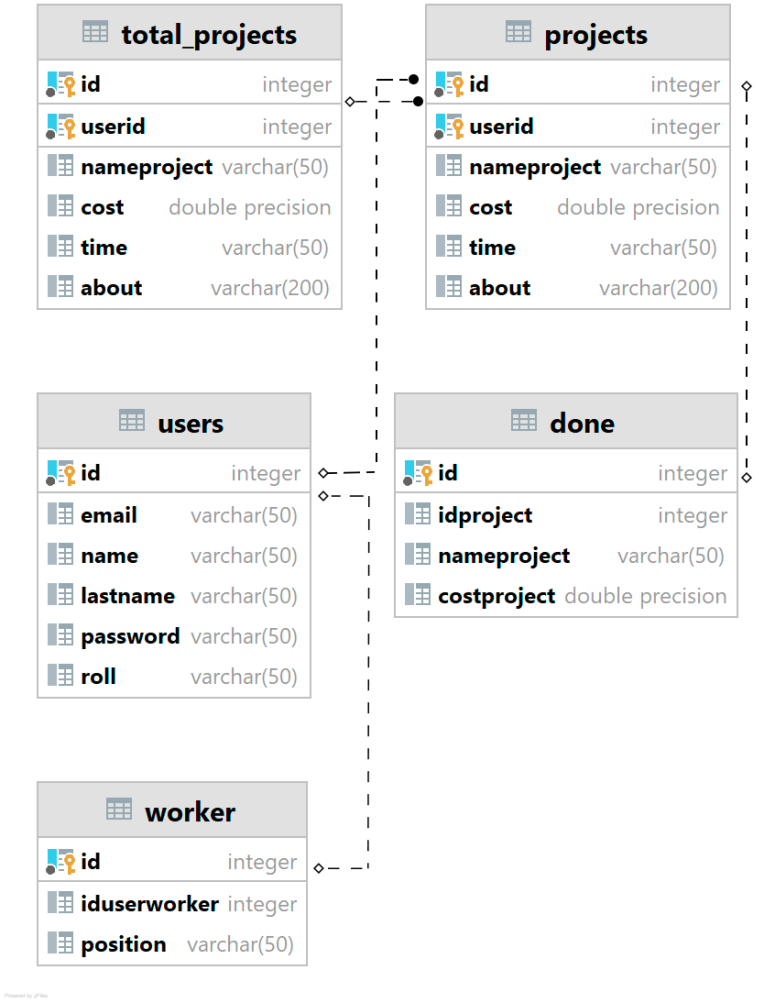

Рисунок 4.1 –Модель системы

Установленные стандарты позволяют избежать различной трактовки построенной модели. Данная информационная модель была успешно приведена к третьей нормальной форме, где каждый не ключевой атрибут не транзитивно зависит от первичного ключа.

5 МОДЕЛИ ПРЕДСТАВЛЕНИЯ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

5.1 Диаграмма вариантов использования (use case diagram)

UML — это инструментарий моделирования, который используется для построения диаграмм.

Диаграммы прецедентов UML идеально подходят для:

- представление целей взаимодействия системы с пользователем;
- определения и организации функциональных требований в системе;
- указания контекста и требований системы;
- моделирования основного потока событий в сценарии использования.

Сценарий — это чёткая последовательность действий, которая показывает поведение. При разработке пользовательского интерфейса он описывает взаимодействие между пользователем (или категорией пользователей, например, администраторами системы, конечными пользователями) и системой. Такой сценарий состоит из последовательного описания комбинаций отдельных действий и задач (например, нажатий клавиш, щелчков по элементам управления, ввода данных в соответствующие поля и т. д.).

На диаграммах UML актёры изображаются в виде стилизованных человечков. Актеры — это пользователи (может быть человек, организация или внешняя система), которые взаимодействуют с системой.

Варианты использования представлены с помеченной овальной формой. Рисунки-палочки представляют актеров в процессе, а участие актера в системе моделируется линией между актером и сценарием использования. Граница системы рисуется с помощью рамки вокруг самого варианта использования.

Диаграмма прецедентов используется для просмотра поведения системы таким образом, чтобы:

- пользователь мог понять, как использовать каждый элемент;
- разработчик мог реализовать эти элементы.

Графическое представление диаграммы представлено на рисунке 5.1.

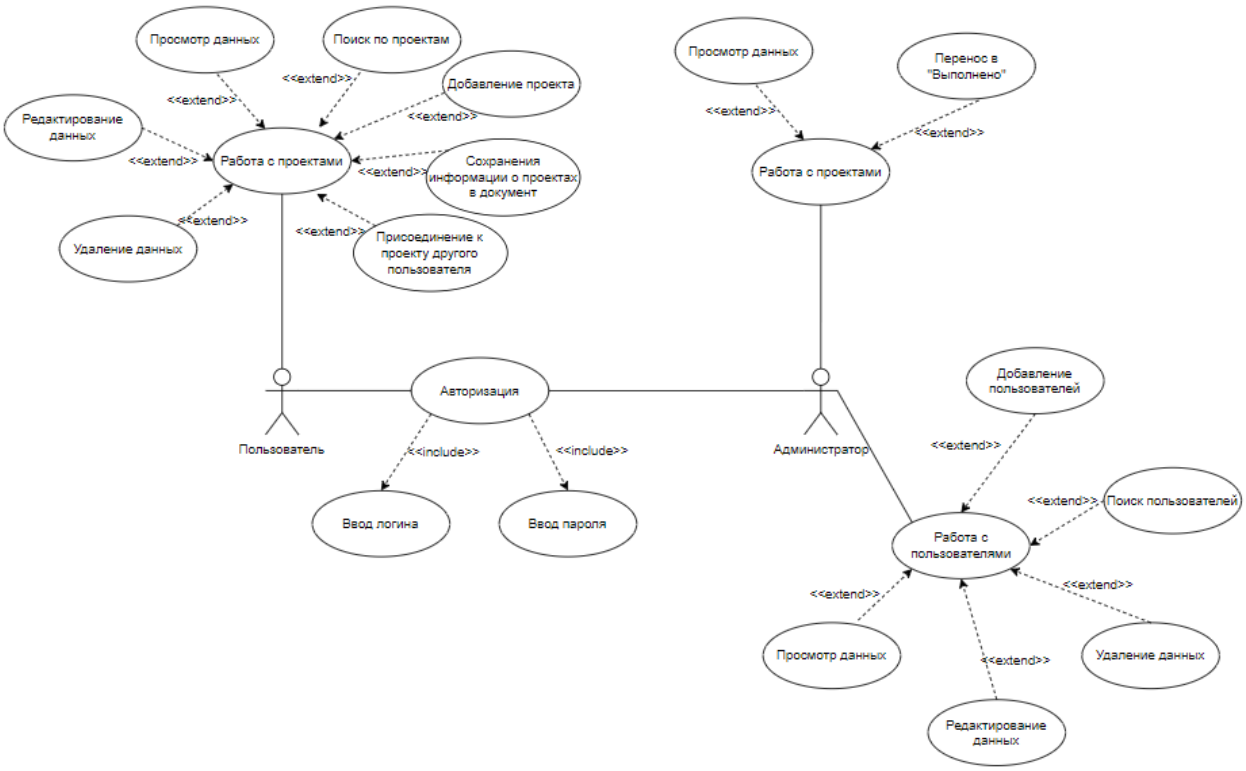

Рисунок 5.1 – Диаграмма вариантов использования

5.2 Диаграмма состояний (statechart diagram)

Диаграмма состояний — это диаграмма, которая используется для описания поведения системы с учетом всех возможных состояний объекта при возникновении события. Это поведение представлено и проанализировано в серии событий, которые происходят в одном или нескольких возможных состояниях. Каждая диаграмма представляет объекты и отслеживает различные состояния этих объектов по всей системе.

Каждая диаграмма состояний обычно имеет начало — темный круг, который обозначает начальное состояние, и конец— круг с рамкой, который обозначает конечное состояние. Но несмотря на наличие четких начальной и конечной точек, диаграммы состояний не обязательно являются лучшим инструментом для отслеживания общего развития событий. Скорее, они иллюстрируют конкретные виды поведения - в частности, переходы из одного состояния в другое.

Диаграммы состояний в основном изображают состояния и переходы. Состояния представлены прямоугольниками с закругленными углами. Переходы отмечены стрелками, которые переходят из одного состояния в другое, показывая, как изменяются состояния.

Графическое представление диаграммы состояний представлено на рисунке 5.2.

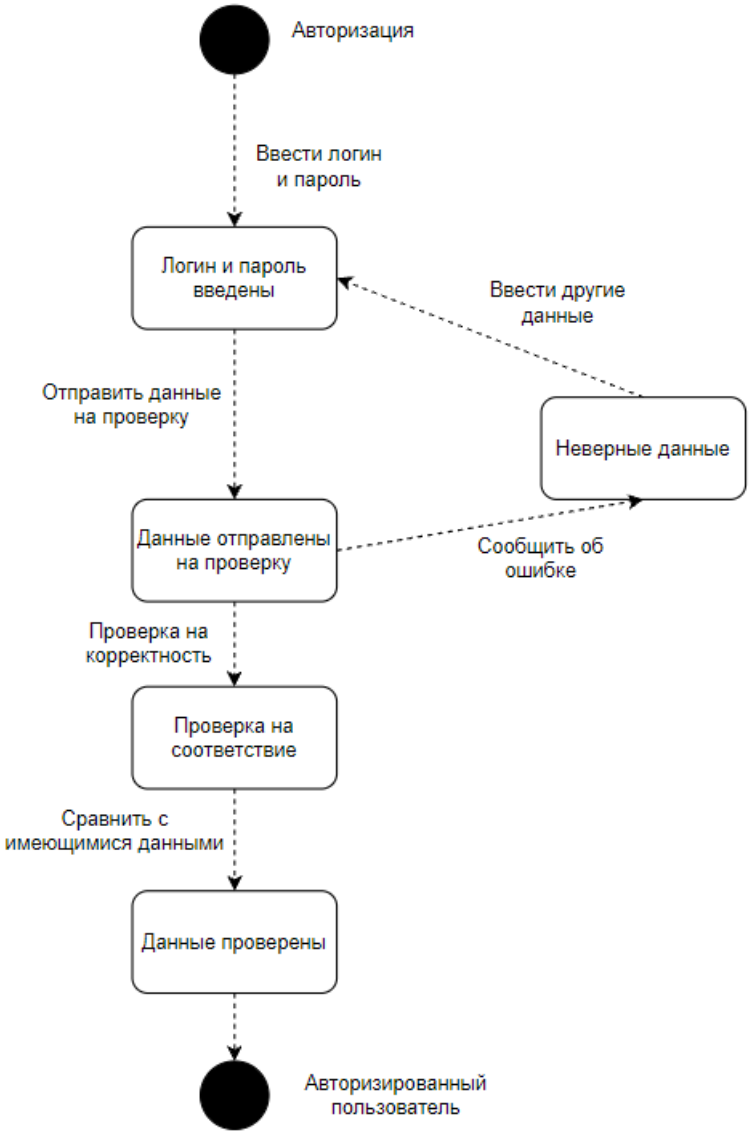

Рисунок 5.2 – Диаграмма состояний

5.3 Диаграмма последовательностей (Sequence diagram)

Диаграммы последовательности иногда называют диаграммами событий или сценариями событий, они описывают, как и в каком порядке группа объектов взаимодействуют. Эти диаграммы используются разработчиками программного обеспечения и бизнес-профессионалами для понимания требований к новой системе или для документирования существующего процесса.

Символ объекта представляет класс или объект в UML и демонстрирует, как объект будет вести себя в контексте системы. Атрибуты класса не должны быть перечислены в этой форме.

Коробка активации представляет время, необходимое объекту для выполнения задачи. Чем дольше будет выполняться задание, тем дольше будет окно активации.

Символ актера показывает объекты, которые взаимодействуют или являются внешними по отношению к системе.

С помощью стрелок и символов сообщения информация передается между объектами. Эти символы могут отражать начало и выполнение операции или отправку и прием сигнала.

Синхронный символ сообщения — это сплошная линия со сплошной стрелкой. Этот символ используется, когда отправитель должен дождаться ответа на сообщение, прежде чем оно продолжится. На диаграмме должны отображаться как звонок, так и ответ.

Асинхронный символ сообщения — это сплошная линия с подкладкой стрелки. Асинхронные сообщения не требуют ответа перед продолжением отправителя. Только диаграмма должна быть включена в диаграмму.

Символ асинхронного обратного сообщения представлен пунктирной линией с подкладкой стрелки.

Асинхронный символ создания сообщения представлен пунктирной линией с подкладкой стрелки. Это сообщение создает новый объект.

Символ ответного сообщения — это пунктирная линия со стрелкой на линии.

Удалить символ сообщения представлен сплошной линией со сплошной стрелкой, за которой следует X. Это сообщение уничтожает объект.

Графическое представление диаграммы последовательностей представлено на рисунке 5.3.

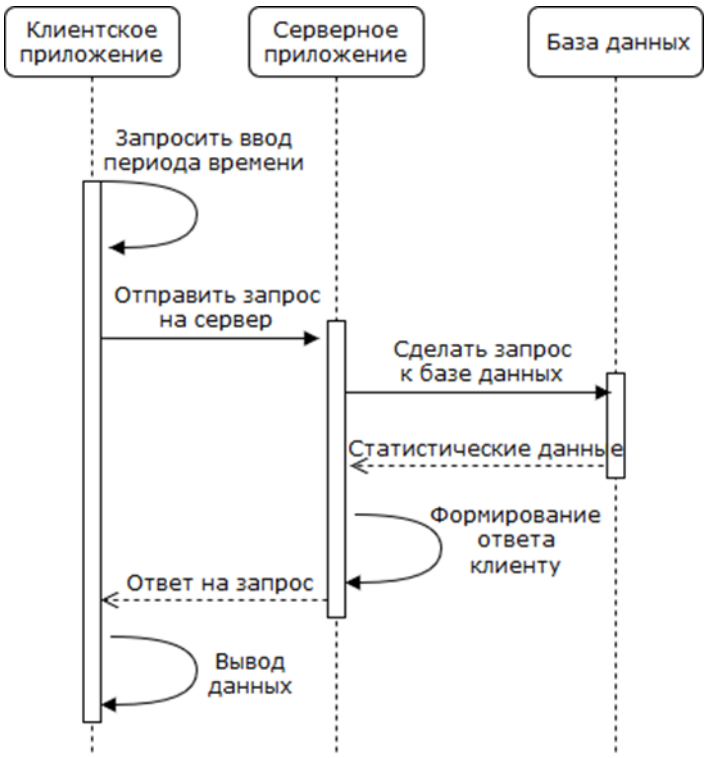

Рисунок 5.3 – Диаграмма последовательности

5.4 Диаграмма компонентов (component diagram)

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

Компонент (component) — элемент модели, представляющий некоторую модульную часть системы с инкапсулированным содержимым, спецификация которого является взаимозаменяемой в его окружении.

Графическое представление диаграммы последовательностей представлено на рисунке 5.4.

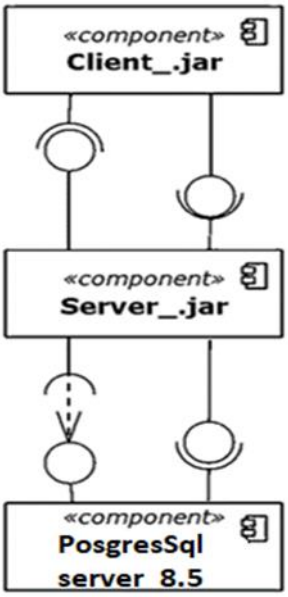

Рисунок 5.4 – Диаграмма компонентов

5.5 Диаграмма развертывания (deployment diagram)

Диаграмма развёртывания – один из доступных видов диаграмм, поддерживаемых Flexberry.

Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого и нужны диаграммы развёртывания, которые иногда называют диаграммами размещения.

Графическое представление диаграммы последовательностей представлено на рисунке 5.5.

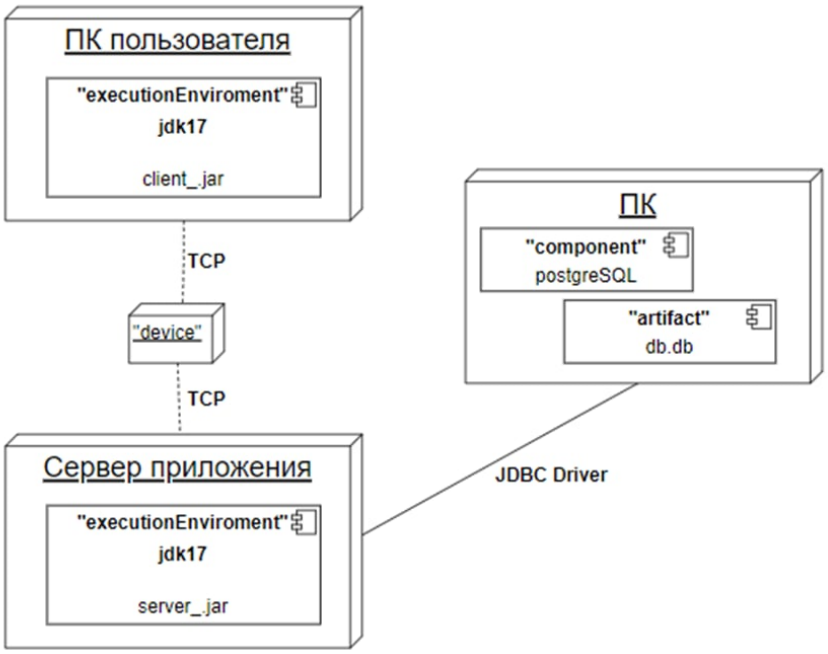

Рисунок 5.5 – Диаграмма развертывания

5.6 Диаграмма классов (static structure diagram)

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Целью создания диаграммы классов является графическое представление статической структуры декларативных элементов системы (классов, типов и т. п.) Она содержит в себе также некоторые элементы поведения (например — операции), однако их динамика должна быть отражена на диаграммах других видов (диаграммах коммуникации, диаграммах состояний). Для удобства восприятия диаграмму классов можно также дополнить представлением пакетов, включая вложенные.

При представлении сущностей реального мира разработчику требуется отразить их текущее состояние, их поведение и их взаимные отношения. На каждом этапе осуществляется абстрагирование от маловажных деталей и концепций, которые не относятся к реальности (производительность, инкапсуляция, видимость и т. п.).

Графическое представление диаграммы последовательностей представлено на рисунке 5.6.

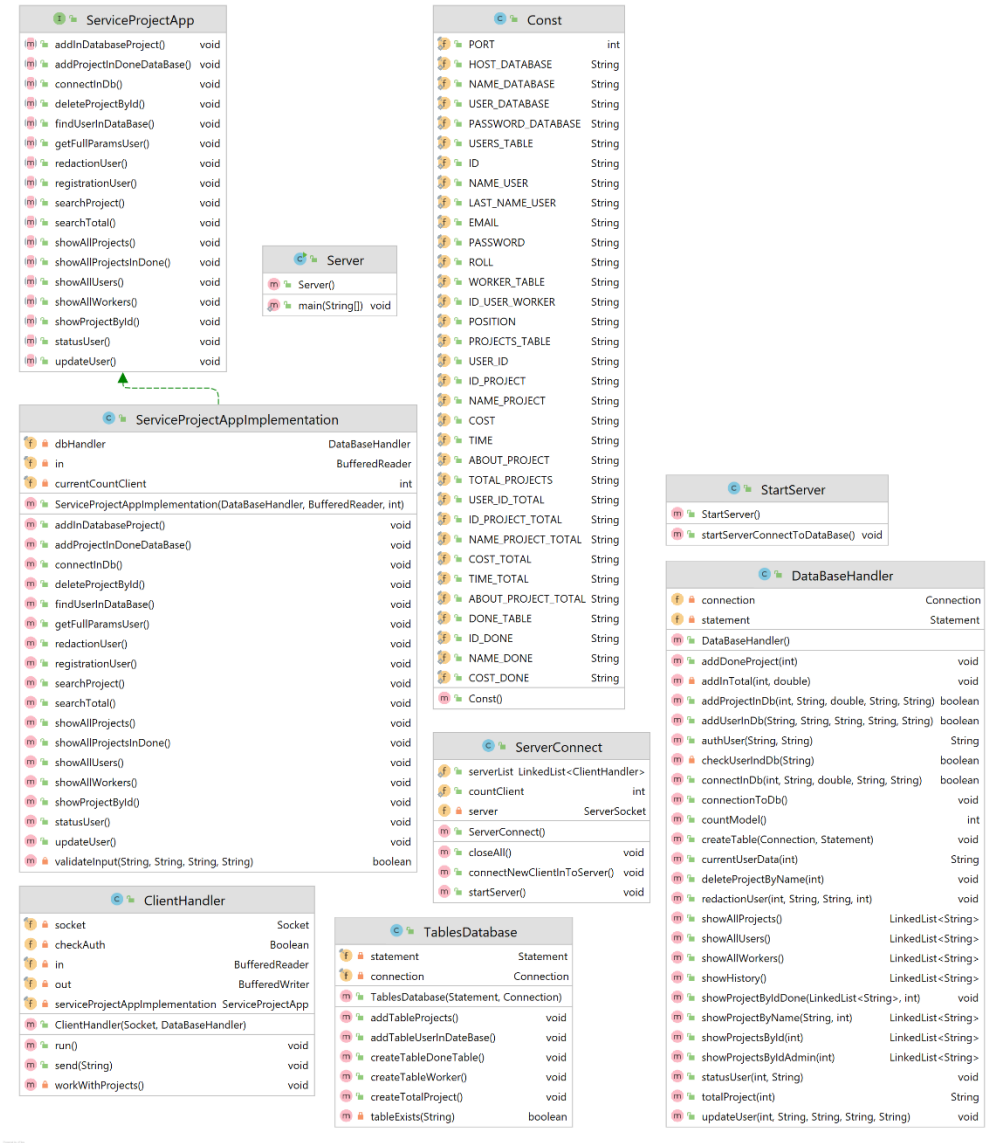

Рисунок 5.6 – Диаграмма классов

6 ОПИСАНИЕ АЛГОРИТМОВ, РЕАЛИЗУЮЩИХ БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

6.1 Алгоритм работы функции обработки входящего запроса

На сервере обработка входящих запросов реализована следующим образом: имеется функция, получающая входящие запросы, которые на основе данных запроса получает экземпляр класса, обрабатывающего запрос. Далее производится вызов метода обрабатывающего класса. Внутри данного метода производятся необходимые действия на основе запроса и возвращает результат, который далее передаётся клиенту.

Алгоритм представлен на рисунке 6.1.

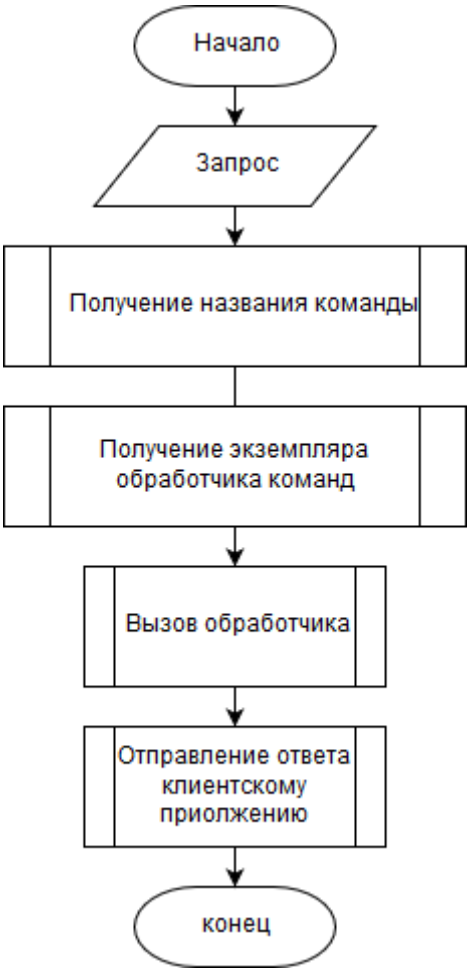

Рисунок 6.1 – Алгоритм работы функции обработки входящего запроса

6.2 Алгоритм работы функции поиска по проектам

Имеется возможность осуществлять поиск по следующим сущностям: проект. Для этого необходимо в клиентском приложении в соответствующей сущности нажать кнопку «Поиск». В окне необходимо ввести данные, на основе которых будет осуществляться поиск. Далее, по нажатию на кнопку «Поиск», будет отправлен соответствующий запрос на сервер, который в свою очередь сделает запрос к базе данных, получит ID сущностей, удовлетворяющих поисковому запросу, и вернёт их клиентскому приложению. Найденные сущности будут выведены в таблице.

Алгоритм представлен на рисунке 6.2.

Рисунок 6.2 – Алгоритм работы функции поиска по арендным платежам

7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ЧАСТИ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

Данное программное обеспечение представляет собой систему проектного менеджмента и программной поддержки управления сроками и стоимостью проекта. Чтобы войти в систему нужно воспользоваться окном авторизации пользователей, которое продемонстрировано на рисунке 7.1.

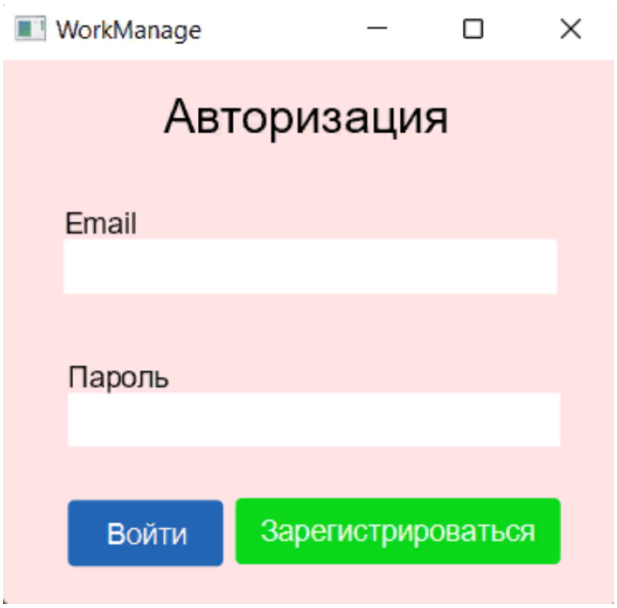

Рисунок 7.1 – Окно авторизации пользователей системы

После авторизации открывается главное окно выбранного пользователя для дальнейшей работы в системе. Рассмотрим сначала вариант работы программы для администратора. Он имеет доступ ко всем проектам.

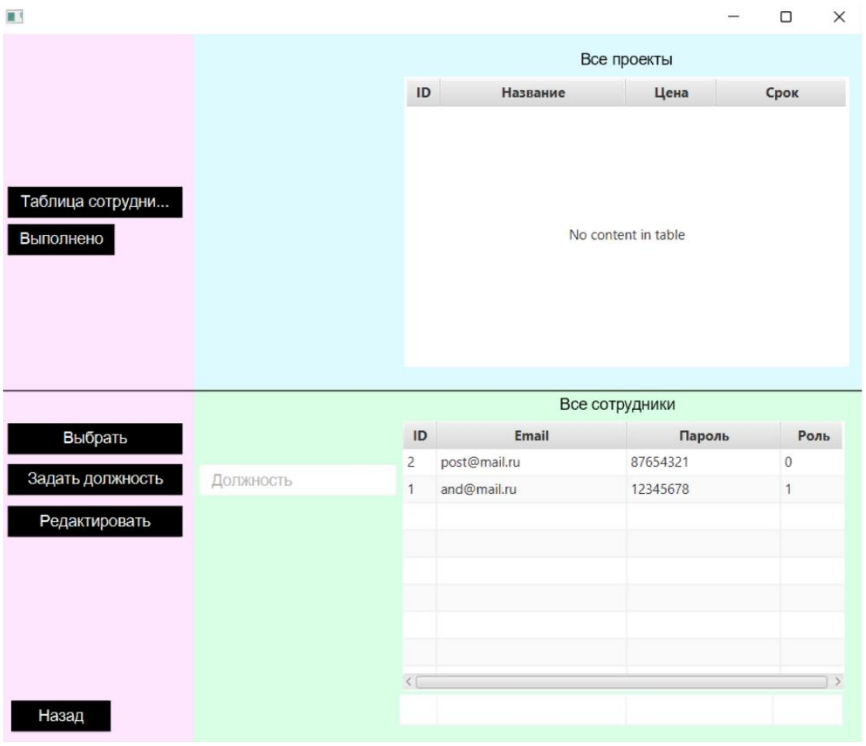

Рисунок 7.2 – Главное окно администратора системы

При переходе на вкладку «Сотрудники» администратор видит информацию о том или ином сотруднике. Так же имеется возможность добавления и редактирования сотрудника. Данное окно представлено на рисунке 7.3.

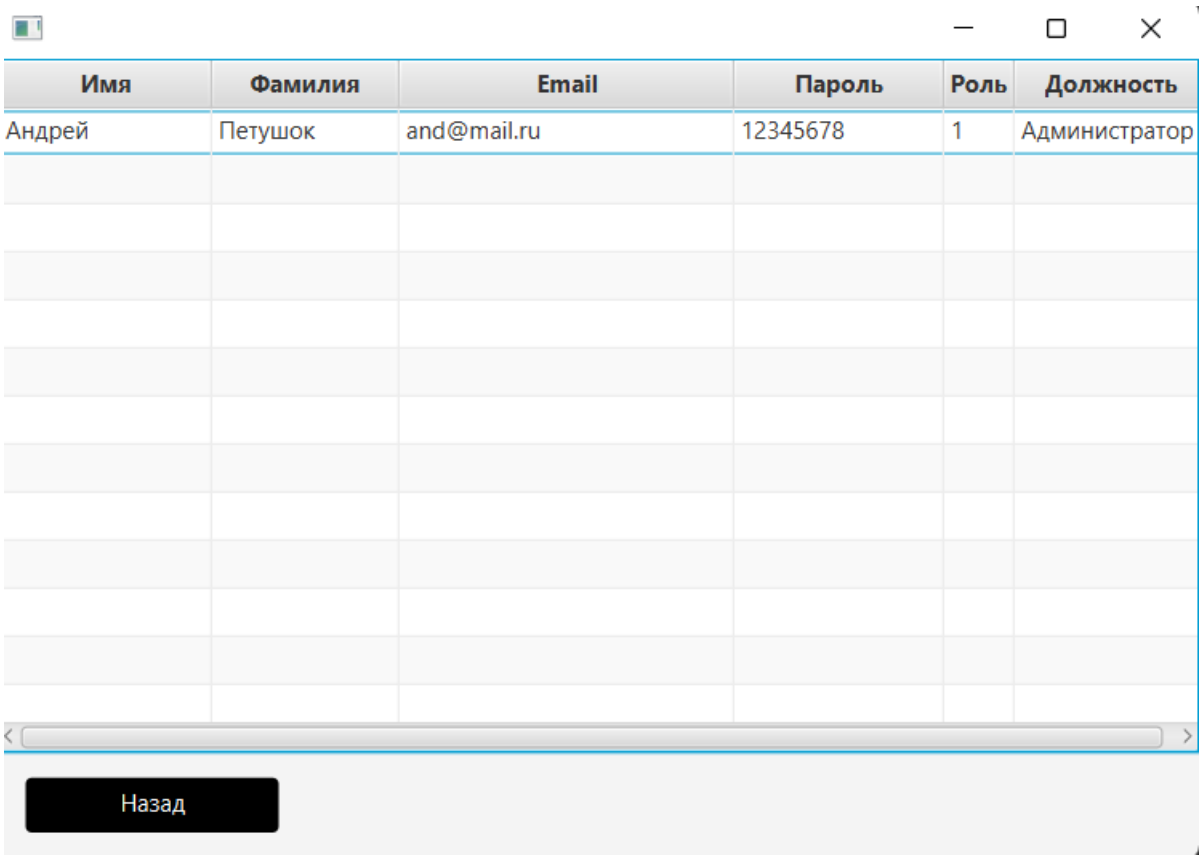

Рисунок 7.3 – Вкладка «Работники»

Администратор имеет возможность редактировать всех пользователей (рисунок 7.4).

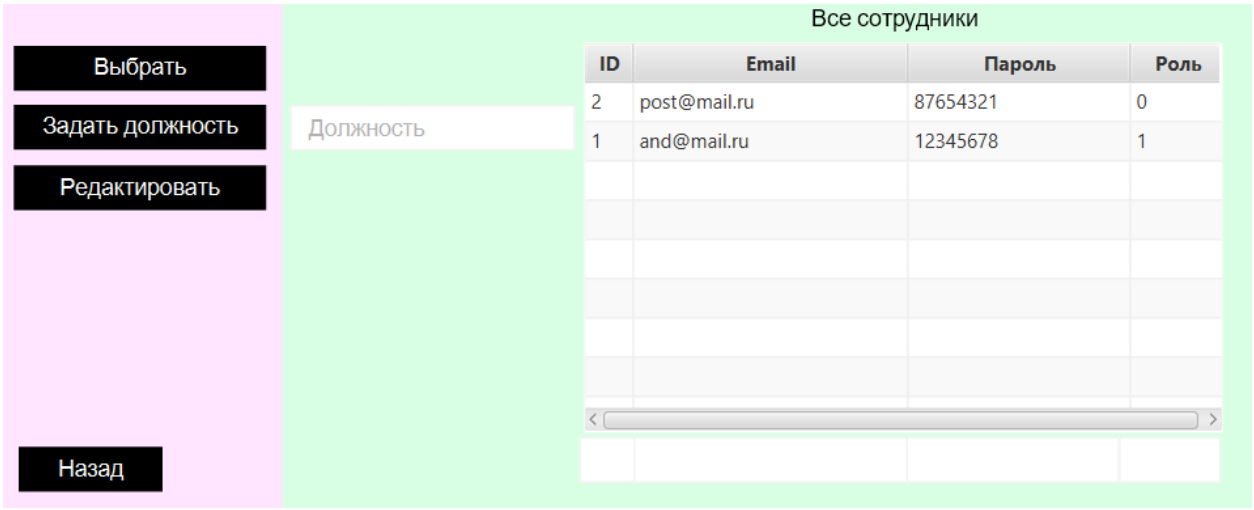

Рисунок 7.4 – Окно всех сотрудников

Вкладка «Проекты» содержит в себе информацию о названии проекта, его стоимости и сроках. Имеется возможность удаления, добавления новой информации, присоединения к уже имеющемуся проекту (рисунок 7.5).

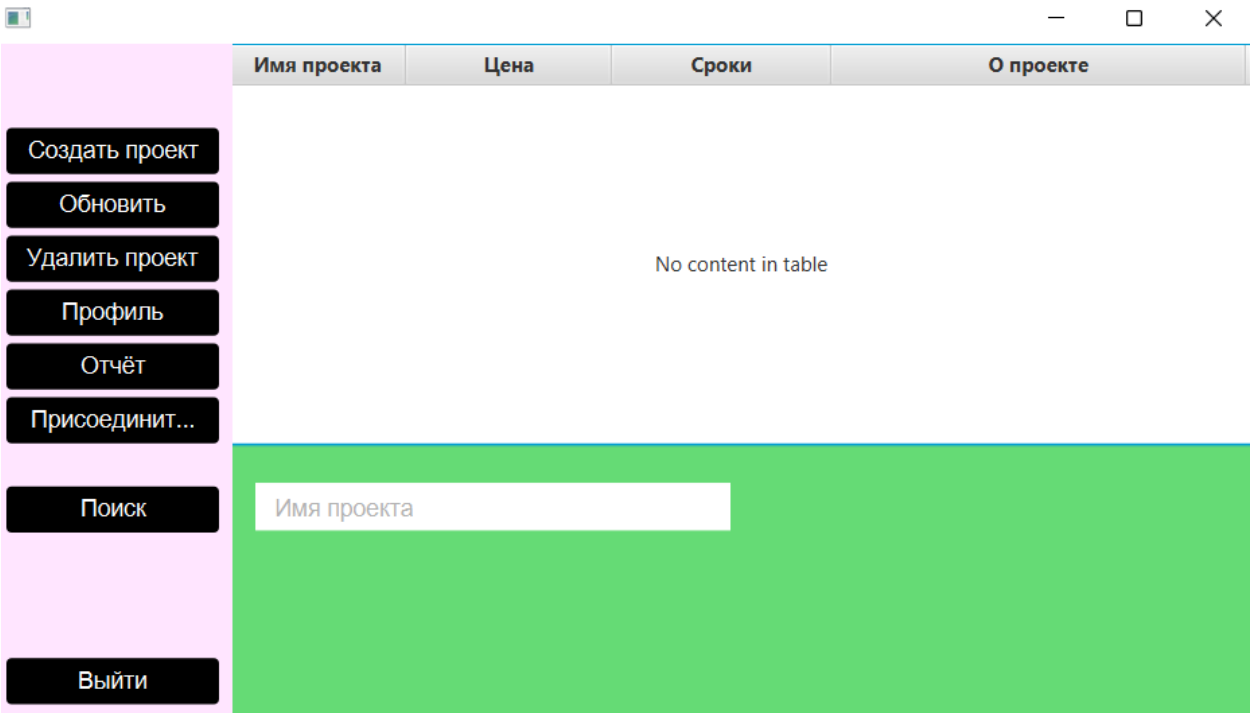

Рисунок 7.5 – Вкладка «Пользователя»

Вкладка «Присоединиться» подразумевает в себе окно с таблицей всех имеющихся проектов. Пользователь может выбрать из имеющихся проектов тот, в котором он будет участвовать. После этого выбранный проект добавится в таблицу проектов сотрудника (рисунок 7.6).

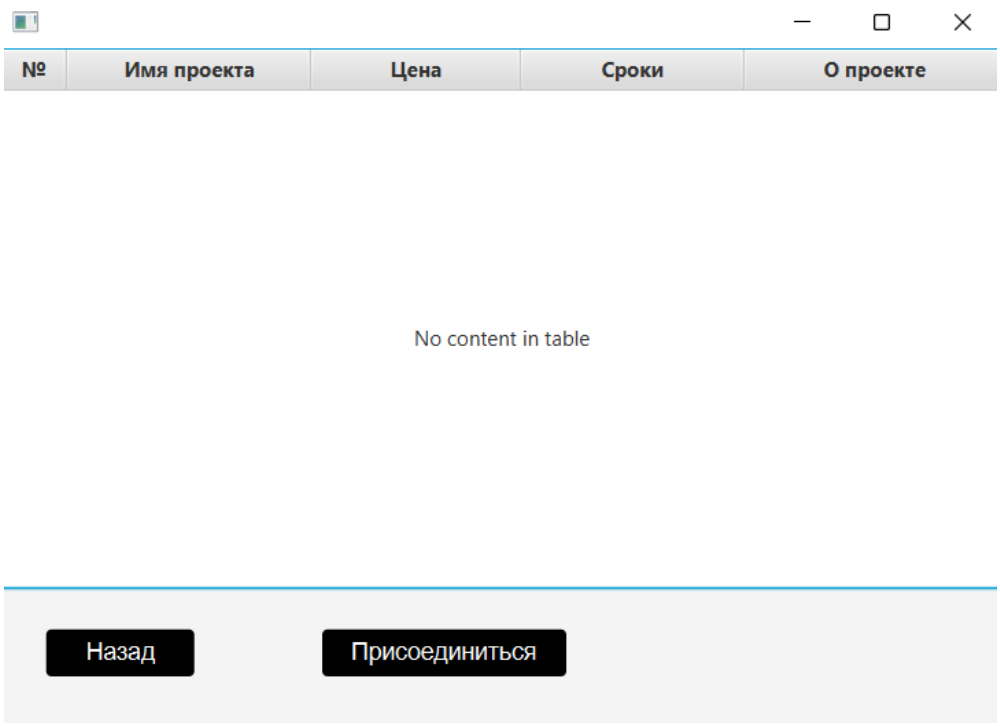

Рисунок 7.6 – Вкладка «Присоединиться»

Так же в данном окне есть возможность редактировать учетную запись текущего юзера (рисунок 7.7)

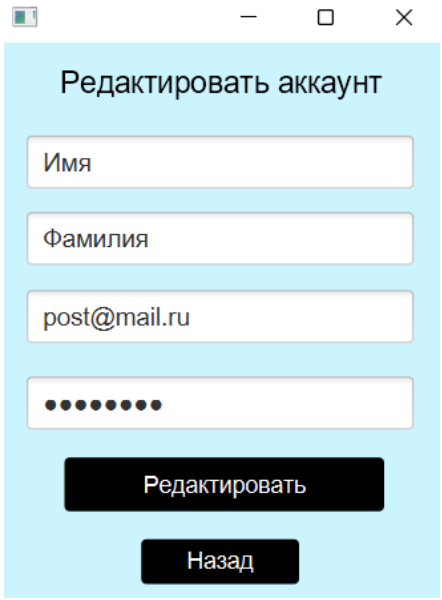

Рисунок 7.7 – Окно изменения учетной записи

8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ ПРОЕКТНОГО МЕНЕДЖМЕНТА И ПРОГРАММНОЙ ПОДДЕРЖКИ УПРАВЛЕНИЯ СРОКАМИ И СТОИМОСТЬЮ ПРОЕКТА

Тестирование систем – важный этап производства ПО, направленный на детальное изучение программного кода и выявление багов в работе продукта.

Моделируя различные ситуации, было выполнено тестирование созданного программного обеспечения с целью:

– того, чтобы приложение в будущем работало правильно при любых обстоятельствах;

– предоставления актуальной информации о состоянии продукта на данный момент;

– того, чтобы конечный продукт соответствовал всем описанным требованиям.

Были выявлены различные исключительные ситуации и предусмотрена их обработка.

В случае если пользователь не выбрал в таблице строку с проектом выводится такое предупреждение (рисунок 8.1).

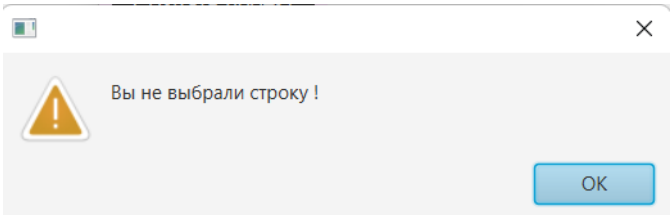

Рисунок 8.1 – Окно уведомления об ошибке «Строка не выбрана»

Также предусмотрена обработка такой ошибки, как пустое значение поля поиска (рисунок 8.2).

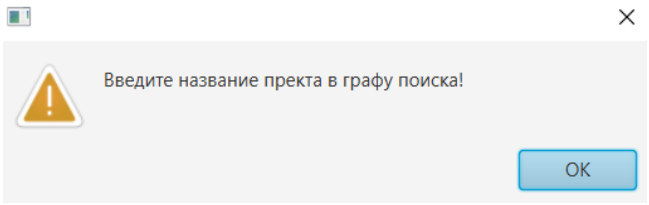

Рисунок 8.2 – Окно уведомления об ошибке «Пустое значение поля поиск»

ЗАКЛЮЧЕНИЕ

Проектный менеджмент – это один из самых прогрессивных методов управления в компании. Его применяют для решения задач в любой сфере: бизнесе, общественной деятельности или госуправлении. Проектный подход позволяет заранее обозначить важные цели и максимально эффективно использовать бюджет и другие ресурсы. Однако сама по себе система не решит проблем. Чтобы выстроить управление в соответствии со спецификой и задачами конкретного бизнеса нужен грамотный менеджер.

В данном курсовом проекте была разработана система проектного менеджмента «WorkManage». В ней предусмотрена возможность авторизации как администратор или пользователь, регистрация новых пользователей. Пользователь может управлять своими проектами, присоединяться к новым, редактировать свои персональные данные. Предусмотрен функционал, упрощающий работу с поиском конкретных проектов и сортировкой.

Администратор в свою очередь может после проверки готового проекта отправлять его в «Выполненные», если он соответствует всем требованиям, прописанным в ТЗ. У Администратора есть возможность видеть всех пользователей программы, редактировать их профиль и назначать должности.

Прототипами данной программы были такие системы как Битрикс24, Jira и других программы, связанные с проектным менеджментом. Был изучен их функционал, выделены основные моменты. После анализа данных продуктов был разработан план реализации программы «WorkManage». Конечно, не все функции были реализованы, но была проделана работа и был встроен необходимый минимум, для комфортной работы в области программного менеджмента.

Предусмотрена возможность задавать проектам сроки и стоимость. Программным методом это вычислить очень сложно, все ключевые моменты оговариваются с заказчиком, к которому нужен свой подход и с которым вы будете работать на определённых условиях. Эти процессы были изображены на диаграмме IDEF0. Поэтому в программу «WorkManage» вносятся только итоговые результаты оценки сроков реализации программного продукта и его стоимости.

Программа «WorkManage» подойдёт для небольших фирм, у которых ещё нет возможности брать крупные заказы. Но для маленьких проектов её функционала вполне хватит. Она упростит учёт проектов и поможет структурировать работу сотрудников.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Эккель, Б. Философия JAVA : учеб. пособие / Б. Эккель – Санкт-Петербург: Питер, 2016. – 1168 с.

[2] Шилдт, Герберт Java 8. Руководство для начинающих: учеб. пособие / Герберт Шилдт – М.: ООО «И.Д. Вильямс», 2015. – 720 с.

[3] Иванов, Д. Моделирование на UML / Д. Иванов, Ф. Новиков – Санкт-Петербург: СПбГУ ИТМО, 2010. – 200 с.

[4] Методология IDEF0 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.expeducation.ru/ru/article/view?id=11003.

[7] Паттерны [Электронный ресурс]. – Электронные данные. – Режим доступа: https://studfiles.net/preview/3640855/.

[8] Java [Электронный ресурс]. – Электронные данные. – Режим доступа: https://metanit.com/java/.

[9] Основы проектного менеджмента [Электронный ресурс]. – Электронные данные. –Режим доступа: http://lms.tpu.ru/pluginfile.php/42670/mod_resource/content/0/IKT/g5/GLAVA_5.pdf/.

ПРИЛОЖЕНИЕ А
(рекомендуемое)
Антиплагиат

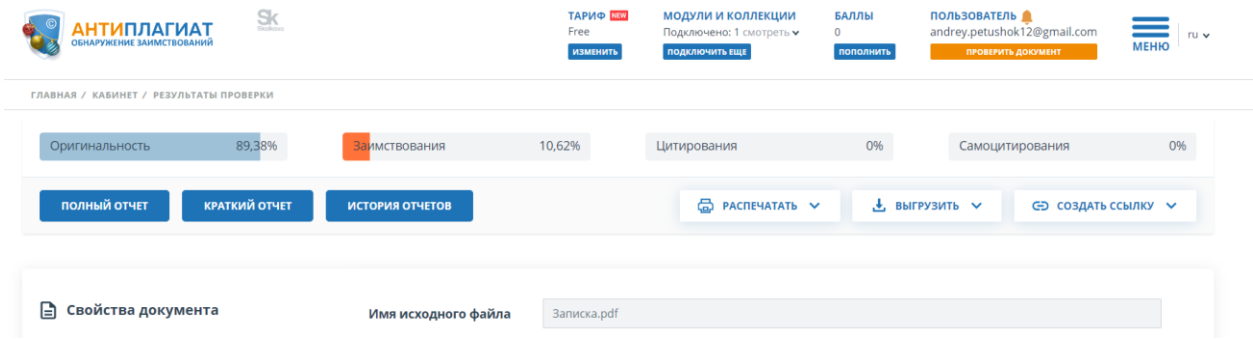

Рисунок А.1 – Антиплагиат

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг алгоритмов, реализующих бизнес-логику

Элементы класса Project:

```
package sample.petushok.model;

public class Project {

    private int id;
    private String nameProject;
    private double cost;
    private String time;
    private String about;

    public Project(int id, String nameProject, double cost, String time, String about) {
        this.id = id;
        this.nameProject = nameProject;
        this.cost = cost;
        this.time = time;
        this.about = about;
    }

    public Project(int id, String nameProject, double cost, String time) {
        this.id = id;
        this.nameProject = nameProject;
        this.cost = cost;
        this.time = time;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNameProject() {
        return nameProject;
    }

    public void setNameProject(String nameProject) {
```

```
        this.nameProject = nameProject;
    }

    public double getCost() {
        return cost;
    }

    public void setCost(double cost) {
        this.cost = cost;
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public String getAbout() {
        return about;
    }

    public void setAbout(String about) {
        this.about = about;
    }

    @Override
    public String toString() {
        return "Project : " +
                "nameProject = " + nameProject +
                ", cost = " + cost +
                ", time = " + time +
                ", about = '" + about + '\"+"'\n";
    }
}
```

Элементы класса User:

```
package sample.petushok.model;

public class User {
```

```
    private String name;
    private String lastName;
    private String password;
    private String email;
    private int roll;
    private String position;

   private int idUser;
   public static User currentUser;

    public User(int idUser,String email, String password, int roll) {
        this.password = password;
        this.email = email;
        this.roll = roll;
        this.idUser = idUser;
    }

    public User(String name, String lastName, String password, String email, int roll, int idUser,
String position) {
        this.name = name;
        this.lastName = lastName;
        this.password = password;
        this.email = email;
        this.roll = roll;
        this.position = position;
        this.idUser = idUser;
    }

    public User(String email, int idUser, int roll) {
     this.email = email;
     this.idUser = idUser;
     this.roll = roll;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public int getRoll() {
        return roll;
    }

    public void setRoll(int roll) {
        this.roll = roll;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getIdUser() {
        return idUser;
    }

    public void setIdUser(int idUser) {
        this.idUser = idUser;
    }
}
```

Основные функции:

```
public class ServiceProjectAppImplementation implements ServiceProjectApp {
    private final DataBaseHandler dbHandler;
    private final BufferedReader in;
    private final int currentCountClient;
    public ServiceProjectAppImplementation(DataBaseHandler dbHandler, BufferedReader in, int currentCountClient) {
        this.dbHandler = dbHandler;
        this.in = in;
        this.currentCountClient = currentCountClient;
    }
    @Override
    public void addProjectInDoneDataBase(){
        try {
            dbHandler.addDoneProject(Integer.parseInt(in.readLine()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void showAllProjectsInDone(){
        LinkedList<String> listDb = new LinkedList<>();
        LinkedList<String> countProject = dbHandler.showHistory();
        int count;
        for(int i=0;i<countProject.size();i++){
            count = Integer.parseInt(countProject.get(i));
            dbHandler.showProjectByIdDone(listDb,count);
        }
        ServerConnect.serverList.get(currentCountClient).send(String.valueOf(listDb.size()));
```

```
        for(int i=0;i<listDb.size();i++){
            System.out.println(listDb.get(i));
            ServerConnect.serverList.get(currentCountClient).send(listDb.get(i));
        }
    }
    @Override
    public void searchTotal(){
        try {
            ServerConnect.serverList.get(currentCountClient).send(
dbHandler.totalProject(Integer.parseInt(in.readLine())));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void redactionUser() {
        String[] subStr;
        try {
            subStr = in.readLine().split(" ");
            dbHandler.redactionUser(Integer.parseInt(subStr[0]),subStr[1],subStr[2],
Integer.parseInt(subStr[3]));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void showAllUsers(){
        LinkedList<String> list = dbHandler.showAllUsers();
        ServerConnect.serverList.get(currentCountClient).send(String.valueOf(list.size()));
        for(String s:list){
```

```
            ServerConnect.serverList.get(currentCountClient).send(s);
        }
    }
    @Override
    public void showAllProjects(){
        LinkedList<String> list = dbHandler.showAllProjects();
        ServerConnect.serverList.get(currentCountClient).send(String.valueOf(list.size()));
        for(String s:list){
            ServerConnect.serverList.get(currentCountClient).send(s);
        }
    }
    @Override
    public void updateUser(){
        String[] str;
        try {
            str=in.readLine().split(" ");
            dbHandler.updateUser(Integer.parseInt(str[0]),str[1],str[2],str[3],str[4]);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void statusUser(){
        String[] subStr;
        try {
            subStr = in.readLine().split(" ");
            dbHandler.statusUser(Integer.parseInt(subStr[0]),subStr[1]);

        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
    @Override
    public void getFullParamsUser(){
        try {
ServerConnect.serverList.get(currentCountClient).send(dbHandler.currentUserData(Integer.pars
eInt(in.readLine())));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void deleteProjectById(){
        try {
            dbHandler.deleteProjectByName(Integer.parseInt(in.readLine()));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void searchProject() throws IOException {
        String[] str;
        str=in.readLine().split(" ");
        LinkedList<String> list = dbHandler.showProjectByName(str[0],Integer.parseInt(str[1]));
        ServerConnect.serverList.get(currentCountClient).send(String.valueOf(list.size()));
        for(String s:list){
            ServerConnect.serverList.get(currentCountClient).send(s);
        }
    }
```

ПРИЛОЖЕНИЕ В
(обязательное)
Листинг основных элементов программы

Подключение к базе данных и запросы:

```
public void connectionToDb(){
      try {
         Class.forName("org.postgresql.Driver");
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
      try {
         connection                                                                    =
DriverManager.getConnection(Const.HOST_DATABASE+Const.NAME_DATABASE,
               Const.USER_DATABASE,
               Const.PASSWORD_DATABASE);
         statement= connection.createStatement();

         System.out.println("Database connection is done");

      } catch (SQLException e) {
         e.printStackTrace();
      }
   }
public boolean addUserInDb(String emailDb, String passwordDb, String rollDb,String
name,String lastName){
        if(!checkUserIndDb(emailDb)) {
           return false;
        }

         try {
            String query = " insert into users (email, password,roll,name,lastName )"
                  + " values ( ?, ?,?,?,?)";

            PreparedStatement preparedStmt = connection.prepareStatement(query);
            preparedStmt.setString (1, emailDb);
            preparedStmt.setString (2, passwordDb);
            preparedStmt.setString (3, rollDb);
            preparedStmt.setString (4, name);
            preparedStmt.setString (5, lastName);


            preparedStmt.executeUpdate();
```

```
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    return true;
}

private boolean checkUserIndDb(String emailUserDb){
    String query = "SELECT " + Const.ID  + " FROM " + Const.USERS_TABLE +
            " WHERE " + Const.EMAIL + " = " + "'" + emailUserDb + "'";
    ResultSet rs = null;
    int idInDb=0;
    try {
        rs = statement.executeQuery(query);
        while (rs.next()) {
          idInDb =  rs.getInt(Const.ID);

        }
        if(idInDb==0){
            return true;
        }

    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
    return false;
}

public String authUser(String email,String password) {
    String currentUser="";
    try {
        String query = "SELECT " + Const.EMAIL + "," +
Const.PASSWORD+","+Const.ID+","+Const.ROLL + " FROM " + Const.USERS_TABLE +
                " WHERE " + Const.EMAIL + " = " + "'" + email + "'" + " AND " +
Const.PASSWORD + " = " + "'" + password + "'";

        ResultSet rs = statement.executeQuery(query);
        while (rs.next()) {
            if (!rs.getString(Const.EMAIL).equals("") &&
                    !rs.getString(Const.PASSWORD).equals("")) {
                currentUser+=rs.getString(Const.ID)+" ";
                currentUser+=rs.getString(Const.ROLL);
            }
        }
```

```
    } catch (SQLException e) {
        e.printStackTrace();
    }

    if(currentUser.equals("")) {
        return "false";
    }
    else {
        return currentUser;
    }
}
```

ПРИЛОЖЕНИЕ Г
(обязательное)
Листинг скрипта генерации баз данных

```
public class TablesDatabase {
    private final Statement statement;
    private final Connection connection;

   public TablesDatabase(Statement statement, Connection connection) {
      this.statement = statement;
      this.connection = connection;

      createTableWorker();
      createTotalProject();
      createTableDoneTable();
      addTableUserInDateBase();
      addTableProjects();
   }

   public void createTableWorker(){

      if(tableExists(Const.WORKER_TABLE)) {
         try {
            String SQL = "CREATE TABLE "+Const.WORKER_TABLE +
                  "( " +
                  " id  SERIAL PRIMARY KEY," +
                  " idUserWorker INTEGER, " +
                  " position VARCHAR (50) " +
                  ")";

            statement.executeUpdate(SQL);
            System.out.println("Таблица  была создана ! " +Const.WORKER_TABLE);
         } catch (SQLException e) {
            e.printStackTrace();
         }
      }
   }

   public void createTotalProject(){
      if(tableExists(Const.TOTAL_PROJECTS)) {
         try {
            String SQL = "CREATE TABLE "+Const.TOTAL_PROJECTS +
                  "( " +
                  " id  SERIAL PRIMARY KEY," +
```

```
                " userId INTEGER ,"+
                " nameProject VARCHAR (50), " +
                " cost DOUBLE PRECISION, " +
                " time VARCHAR (50),"+
                " about VARCHAR (200) "+
                ")";

            statement.executeUpdate(SQL);
            System.out.println("Таблица  была создана ! " +Const.TOTAL_PROJECTS);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

public void createTableDoneTable(){

    if(tableExists(Const.DONE_TABLE)) {
        try {
            String SQL = "CREATE TABLE "+Const.DONE_TABLE +
                "( " +
                " id  SERIAL PRIMARY KEY," +
                " idProject INTEGER, " +
                "nameProject VARCHAR (50)," +
                "costProject DOUBLE PRECISION" +

                ")";

            statement.executeUpdate(SQL);
            System.out.println("Таблица  была создана ! " +Const.DONE_TABLE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

public void addTableUserInDateBase(){
    if(tableExists(Const.USERS_TABLE)) {
        try {
            String SQL = "CREATE TABLE "+Const.USERS_TABLE +
                "( " +
                " id  SERIAL PRIMARY KEY," +
                " email VARCHAR (50), " +
                " name VARCHAR (50), " +
                " lastName VARCHAR (50), " +
```

```
                " password VARCHAR (50), " +
                " roll VARCHAR (50)"+
                ")";

            statement.executeUpdate(SQL);
            System.out.println("Таблица с users была создана ! "+Const.USERS_TABLE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

public void addTableProjects(){
    if(tableExists(Const.PROJECTS_TABLE)) {
        try {
            String SQL = "CREATE TABLE "+Const.PROJECTS_TABLE +
                "( " +
                " id  SERIAL PRIMARY KEY," +
                " userId INTEGER ,"+
                " nameProject VARCHAR (50), " +
                " cost DOUBLE PRECISION, " +
                " time VARCHAR (50),"+
                " about VARCHAR (200) "+

                ")";

            statement.executeUpdate(SQL);
            System.out.println("Таблица с users была создана ! "+Const.PROJECTS_TABLE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

private boolean tableExists(String nameTable){

    try{
        DatabaseMetaData md = connection.getMetaData();
        ResultSet rs = md.getTables(null, null, nameTable, null);
        rs.last();
        return rs.getRow() <= 0;
    }catch(SQLException ignored){

    }
    return true;}}
```

ПРИЛОЖЕНИЕ Д
Ведомость курсового проекта
(обязательное)

Зона	Обозначение	Наименование	Дополнительные, сведения
		Текстовые документы	
	ГУИР КП 1–40 05 01 041 ПЗ	Пояснительная записка	54 с.
		Графические документы	
	ГУИР.506113.041 Д1	Схема алгоритма	Формат А3
	ГУИР.506113.041 Д2	IDEF0-модель процессов предметной области	Формат А4
	ГУИР.506113.041 Д3	Схема алгоритма поиска	Формат А4
	ГУИР.506113.041 Д4	Диаграммы	Формат А4, 2 л.
	ГУИР.506113.041 Д5	Пользовательский графический интерфейс	Формат А4

Перв. примен. ГУИР. ГУИР.506113.041 Д1

Справочный №

ПоКПи

Инв. №

Взам. Инв. №

ПоКПись и

Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата	ГУИР КП 1–40 05 01 041 Д1		
Разраб.		Петушок		20.11.21	Проектный менеджмент и программная поддержка управления сроками и стоимостью проекта Ведомость курсового проекта	Лит.	Лист 54 / Листов 54
Пров.		Унучек		04.12.21		Т	
Н.контр.		Унучек		04.12.21		Кафедра ПИКС, группа 914301	
Утв.		Хорошко					