

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
магистр.эконом.наук, старший
преподаватель
_____ Т. М. Унучек
_____._____.2021

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:
**«УПРАВЛЕНИЕ IT-ПРОЕКТАМИ И РАЗРАБОТКА
АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПОДДЕРЖКИ ПРОЦЕССОВ
СОСТАВЛЕНИЯ ОТЧЁТНОСТИ»**

БГУИР КП 1-40 05 01-10 010 ПЗ

Выполнил студент группы 914301
ГУКОВА Мария Денисовна

(подпись студента)
Курсовой проект представлен на
проверку _____._____.2021

(подпись студента)

Минск 2021

Реферат

БГУИР КР 1-40 05 01-10 010 ПЗ

Гукова М. Д. Управление ИТ-проектами и разработка автоматизированной системы поддержки процессов составления отчётности/
Гукова М. Д. – Минск: БГУИР, 2021. – 60 с.

Пояснительная записка 60 с., 31 рис., 6 источников, 4 приложения

Управление, ИТ-проекты, IntelliJ IDEA, PostgreSQL

Предмет: методы и подходы к управлению ИТ-проектами.

Объект: распределение ИТ-проектов и составление отчетности по ним.

Цель: повышение качества работы ИТ компании, а именно следующих отделов: разработка, тестирование, аналитика, менеджмент.

Методология проведения работы: в процессе решения поставленных задач были рассмотрены принципы работы базы данных, использованы принципы объектно-ориентированного программирования, построение графического интерфейса реализовано с помощью JavaFX.

Результаты работы: при выполнении курсовой работы были рассмотрены основные требования и задачи проектированной системы. Автоматизация управления ИТ-проектами была выполнена в полной мере, что позволяет увеличить эффективность и конкурентоспособность системы. Автоматизация была проведена в основных процессах разработки продукта. Благодаря этому появилась возможность обеспечить комфортное и своевременное взаимодействие с пользователем внутри компании.

Область применения полученных результатов: повседневная жизнь, внутри ИТ- компаний, многие отрасли техники, имеющие отношение к ИТ- сфере.

СОДЕРЖАНИЕ

Введение.....	6
1 Описание предметной области.....	8
2 Постановка задачи и обзор методов её решения.....	10
3 Функциональное моделирование на основе стандарта IDEF0.....	12
3.1 Методология IDEF0.....	12
3.2 Описание разработанной функциональной модели.....	13
4 Информационная модель системы и её описание.....	18
5 Описание алгоритмов, реализующих бизнес-логику серверной части проектируемой системы.....	22
6 Руководство пользователя.....	26
7 Архитектурный шаблон проектирования.....	33
7.1 Разработка диаграммы вариантов использования	34
7.2 Разработка диаграммы состояний	38
7.3 Разработка диаграммы последовательностей	35
7.4 Разработка диаграммы компонентов.....	36
7.5 Разработка диаграммы развёртывания.....	37
7.6 Разработка диаграммы классов	38
8 Результаты тестирования разработанной системы.....	42
Заключение.....	43
Список использованных источников.....	44
Приложение А (обязательное) Листинг основных элементов программы ..	45
Приложение Б (обязательное) Листинг скрипта генерации базы данных.....	53
Приложение В (обязательное) Графический материал	56
Приложение Г (обязательное) Ведомость курсового проекта	60

ВВЕДЕНИЕ

Информационная система есть совокупность технического, программного и организационного обеспечения, а также персонала, предназначенная для того, чтобы своевременно обеспечивать людей надлежащей информацией.

Благодаря использованию информационных технологий, сокращается время на обработку информации, осуществляется хранение больших объемов информации, ускоряется поиск необходимой информации, и выдача ее в удобном для пользователя виде. Именно поэтому применение информационной системы в ИТ-проектах так необходимо. Ведь ИТ-проекты – это место, переполненное данными различного содержания.

В последнее время для автоматизации и оптимизации внутренней деятельности предприятий все чаще применяются новые программные продукты, призванные автоматизировать и формализовать отношения внутри технологических цепочек и, таким образом, сведя к минимуму человеческий фактор в принятии решений, оптимизировать весь процесс управления товарными потоками.

Управление проектами (англ. project management) – область деятельности, в ходе которой определяются и достигаются четкие цели при балансировании объемом работ, ресурсами (такими как время, деньги, труд, материалы, энергия, пространство и др.), временем, качеством и рисками в рамках некоторых проектов, направленных на достижение определенного результата при указанных ограничениях.

Управление проектом – применение знаний, навыков, инструментов и методов для планирования и реализации действий, направленных на достижение поставленной цели в рамках проектных требований.

Управление проектами включает такие этапы, как:

- планирование работ;
- оценка рисков;
- оценка необходимых ресурсов;
- организация работ;
- привлечение людских и материальных ресурсов;
- назначение задач;
- руководство;
- контроль над ходом выполнения (для измерения и контроля эффективности выполнения проектов);
- отчет о ходе выполнения;

- анализ результатов на основе полученных фактов.

Для производительности любого предприятия необходимо наличие развитой системы, которая способна осуществить автоматизированный сбор, хранение, обработку и управление данными. Также данная система должна включать в себя технические средства обработки данных, программное обеспечение и обслуживающий персонал.

Целью курсового проекта является повышение качества работы ИТ компании, что можно достичь автоматизацией управления системы ИТ-компании.

Поставленная цель потребовала решения следующих задач:

- исследовать предметную область;
- проанализировать логическую и физическую модель представления данных;
- создать базу данных, содержащей данные о пользователях, семейных счетах, расходах, доходах, категориях;
- реализовать возможности сортировки и фильтрации записей по определённым критериям;
- реализовать возможности составления статистики о семейном бюджете за определённый период времени;
- реализовать возможности регистрации и авторизации;
- разделить доступа к управлению программой на пользовательский и администраторский с соответствующими программными возможностями;
- разработать удобный и интуитивно понятный пользователю интерфейс;
- сделать вывод о необходимых доработках.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

ИТ-проект – это проект, в рамки которого входят работы, связанные с информационными технологиями.

Информационные технологии – это технологии, направленные на создание, развитие и поддержку информационных систем.

В широком смысле под информационными системами понимают следующее: «информационной системой называется комплекс, включающий вычислительное и коммуникационное оборудование, программное обеспечение, лингвистические средства и информационные ресурсы, а также системный персонал и обеспечивающий поддержку динамической информационной модели некоторой части реального мира для удовлетворения информационных потребностей пользователей.

Управление – это процессы организации, планирования, мотивации и контроля, направленных на достижение целей организации.

Что такое управление проектами? Управление проектами (project management) в широком понимании – это профессиональная деятельность, основанная на использовании современных научных знаний, навыков, методов, средств и технологий и ориентированная на получение эффективных результатов.

Применение методологии управления проектами дает возможность четко определить цели и результаты проекта, дать им количественные характеристики, временные, стоимостные и качественные параметры проекта, создать четкий план проекта, выделить, оценить риски и предотвратить возможные негативные последствия во время реализации проекта.

На сегодняшний день методология управления проектами доказала свое право считаться одним из самых эффективных способов успешной реализации проектов.

Управление проектами – это универсальный язык общения в среде профессионалов любой области.

Универсальные знания и методы управления проектами позволяют решать такие задачи, как:

- определение целей проекта;
- подготовка обоснования проекта;
- его структурирование (подцели, подпроекты, фазы и т.д.);
- определение финансовых потребностей и источников финансирования;

- подбор поставщиков, подрядчиков и других исполнителей (на основе процедур торгов и конкурсов);
- подготовка и заключение контрактов;
- расчет сметы и бюджета проекта;
- определение сроков выполнения проекта и разработка графика реализации;
- контроль за ходом выполнения проекта и внесения корректив в план реализации;
- управление рисками в проекте;
- обеспечение контроля за ходом выполнения проекта.

Управление ИТ-проектами включает надзор за проектами разработки программного обеспечения, установки оборудования, модернизации сетей, развертывания облачных вычислений и виртуализации, проектов бизнес-аналитики и управления данными, а также внедрения ИТ-услуг.

В дополнение к обычным проблемам, которые могут привести к сбою проекта, факторы, которые могут негативно повлиять на успех ИТ-проекта, включают достижения в области технологий во время выполнения проекта, изменения инфраструктуры, которые влияют на безопасность и управление данными, и неизвестные зависимые отношения между оборудованием, программным обеспечением, сетевой инфраструктурой и данными.

2 ПОСТАНОВКА ЗАДАЧИ И МЕТОДЫ ЕЕ РЕШЕНИЯ

Постановка задачи

Целью курсового проекта является оптимизация деятельности ИТ-компании за счет автоматизации процесса учёта и контроля проектов, рисков, ресурсов, а также процесса составления статистики о состоянии готовности проектов и возможных рисках.

Для достижения поставленной цели будет разработано программное обеспечение, которое поможет пользователям быстро искать записи о проектах, составлять отчёт о рисках и дедлайнах и проводить анализ потенциальных заказчиков. Программное обеспечение будет представлено в виде GUI-приложения, предоставляющее соответствующий функционал в зависимости от роли вошедшего пользователя.

Задача данной программы заключается в обеспечении пользователя всей необходимой информацией об интересующей его литературе. Необходимо, чтобы программа была построена на основе клиент-серверной архитектуры.

В приложении необходимо реализовать:

- авторизация/регистрация пользователей (для разделения прав доступа администратору и пользователю);
- пользователь должен видеть всю имеющуюся информация о нужных ему проектах и составлять отчет о рисках;
- администратор должен иметь возможность добавлять, редактировать и удалять уже хранящуюся в базе данных информацию, а также составлять отчёт о уровне готовности проектов.

Обзор методов поставленной задачи

В качестве основной технологии был выбран язык программирования Java, используемый для разработки сетевых приложений, десктопных приложений, веб-приложений.

Java предоставляет несколько библиотек для создания графических пользовательских интерфейсов. В данном курсовом проекте использовалась самая молодая библиотека JavaFX, которая имеет расширенный список элементов пользовательского интерфейса и поддержку медиа. Также преимуществом данной библиотеки является наличие среды разработки SceneBuilder, который можно установить, как отдельное приложение или интегрировать в среду разработки (например, IntelliJIDEA). JavaFX приложения работают по принципу MVC(Model-View-Controller). Этот подход предоставляет удобную работу с данными и представлениями. В качестве представлений выступают fxml-файлы, где располагаются элементы

пользовательского интерфейса. Контроллеры и модели являются java-классами. Модель – сущность, с которой осуществляется работа. Модель может существовать независимо от представления и контроллера. Контроллер содержит логику работы с данными, обработку событий, происходящих в представлении и вывод данных.

Для реализации клиент-серверного соединения использовался протокол TCP/IP, поскольку он предоставляет более надёжное соединение, обеспечивающее целостность передаваемых данных.

В качестве используемой базы данных был выбран PostgreSQL. Даная база данных полностью соответствует требованиям системы, поскольку принцип её работы основан на ACID-принципах. А – элементарность (atomicity): этот принцип требует, чтобы транзакция была либо выполнена, либо отменена (аварийно прекращена). В любом из этих двух случаев данные должны быть правильно сохранены в конечном состоянии, либо в начальном в случае аварии. С – последовательность (consistency): транзакция должна быть максимально простой и давать предсказуемый результат. I – изолированность (isolation): независимость параллельных транзакций. D – устойчивость (durability): нет возможности прервать транзакцию и потерять данные.

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

3 ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ НА ОСНОВЕ СТАНДАРТА IDEF0

3.1 Методология IDEF0

Описание системы с помощью IDEF0 называется функциональной моделью. Функциональная модель предназначена для описания существующих бизнес-процессов, в котором используются как естественный, так и графический языки. Для передачи информации о конкретной системе источником графического языка является сама методология IDEF0.

Методология IDEF0 предписывает построение иерархической системы диаграмм единичных описаний фрагментов системы.

Данная методология предполагает разработку нескольких диаграмм, с помощью которых описываются функция или процесс: контекстная диаграмма, диаграмма верхнего уровня; набор дочерних диаграмм, на которых отражено более детальное представление об объекте моделирования.

Контекстная диаграмма представляет собой один блок со стрелками, которые отражают связи описываемого процесса с внешней средой. Таким образом, можно говорить о том, что контекстная диаграмма показывает область моделирования и ее границы.

Функция, показанная на контекстной диаграмме, раскладывается на подфункции посредством создания данной дочерней диаграммы верхнего уровня. Затем каждая из представленных подфункций раскрывается в виде дочерних диаграмм более низкого уровня. Степень декомпозиции в данном случае определяется целью моделирования, т.е. разбиение функции на подфункции, операции и т.д. происходит до тех пор, пока не будут достигнуты цели и задачи моделирования.

Каждая IDEF0-диаграмма содержит блоки и стрелки. Блоки изображают функции моделируемой системы. Стрелки связывают блоки вместе и отображают взаимодействия и взаимосвязи между ними.

Функциональные блоки (работы) на диаграммах изображаются прямоугольниками, означающими поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты.

IDEF0 – это достаточно простой и наглядный язык описания бизнес-процессов. Стандарт является универсальным, так как используется, как и разработчиками, так и заказчиками для передачи данных.

На сегодняшний день существуют такие инструменты для работы с IDEF0, как BPWin, ERWin, Ramus и многие другие.

3.2 Описание разработанной функциональной модели

Входными данными являются: проекты, риски и пользовательские данные. Результатом работы программы будет являться полученная отчетность и изменения данных в БД. В качестве механизма выступают пользователь и администратор. Управлением будут являться документации и требования валидации БД.

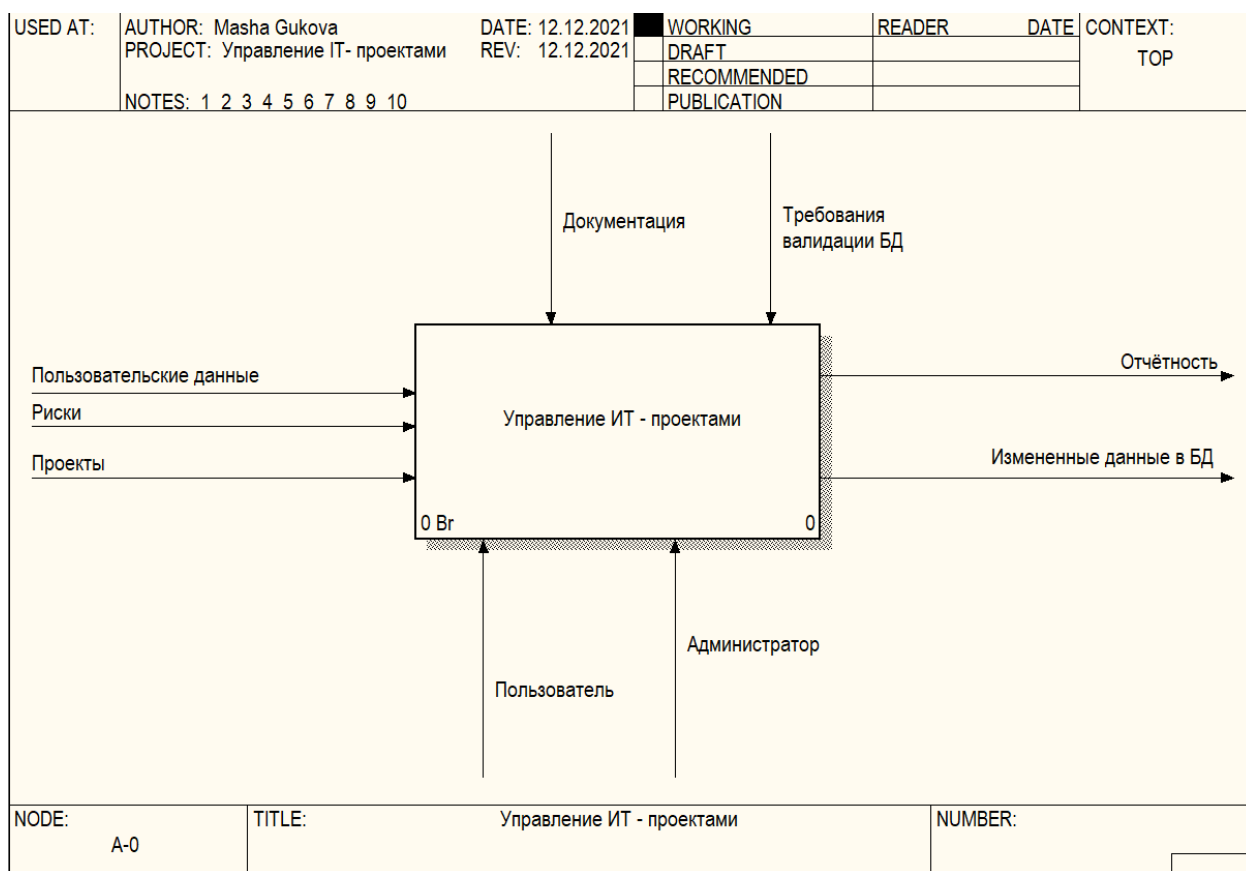


Рисунок 1.1 – Контекстная диаграмма функциональной модели

Как упоминалось ранее, для полного определения перечня вопросов и структуры необходимо провести декомпозицию главного процесса. Для приобретения более чётких и подробных описаний проектируемой системы необходимо осуществить декомпозицию верхнего уровня диаграммы (рисунок 1.2). Особое внимание следует уделить при моделировании данного уровня диаграмм связыванию функций стрелками для решения обозначенных в цели моделирования задач.

Следующая диаграмма отображает основные цели, преследуемые системой. Так как автоматизация позволяет повысить производительность труда, улучшить качество продукции, оптимизировать процессы управления, следовательно необходимо автоматизировать основные процессы.

Данный уровень разбит на 4 функциональных блока: войти в аккаунт, выбрать таблицу, просмотреть аналитику, создать отчёт.

Входные стрелки «пользовательские данные» входят в первый функциональный блок. При удовлетворении валидации, определяются права и роль, в зависимости от которых будет определен доступ к таблицам. Во 2 блоке администратор выбирает таблицу и производит некоторые действия: просматривает, добавляет, удаляет, редактирует и производит поиск. После чего можно создать отчётность по некоторым предложенным критериям.

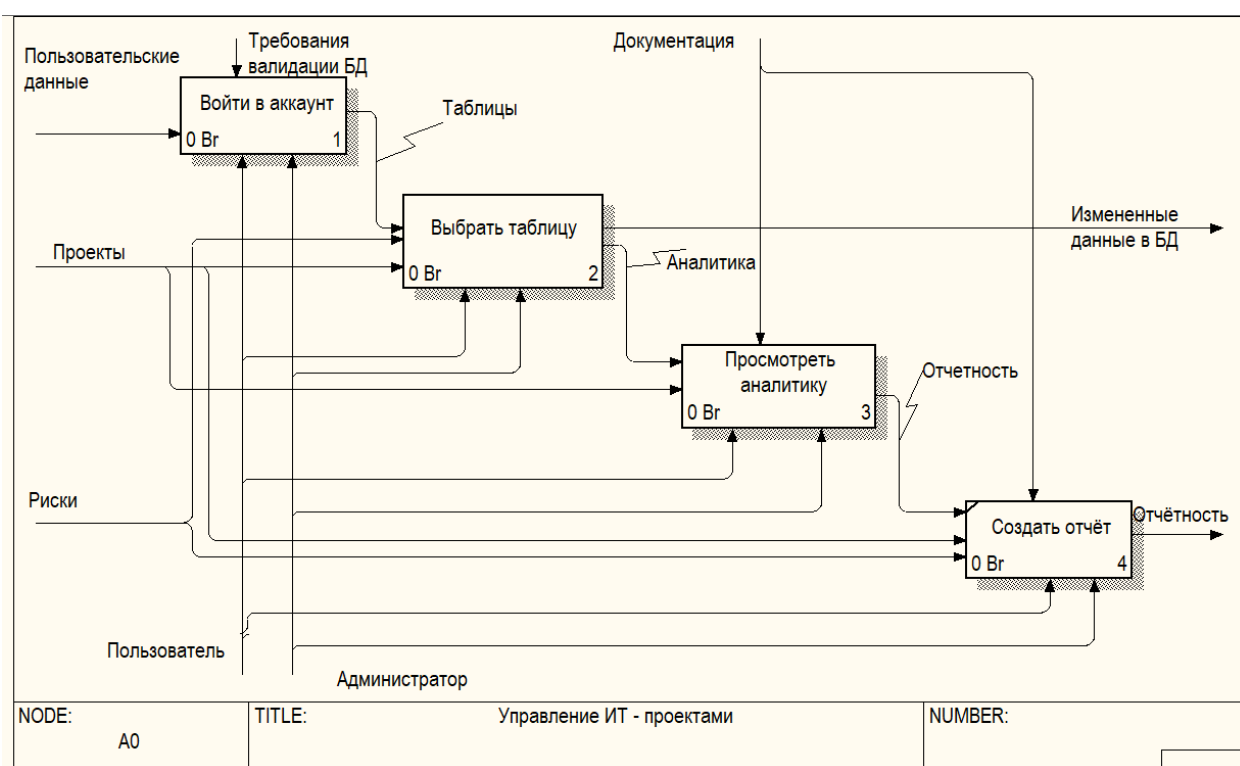


Рисунок 1.2 – Декомпозиция контекстной диаграммы

Рассмотрим третий уровень декомпозиции IDEF0 (см. рисунок 1.3).

Диаграмма представлена двумя блоками: «Вести логин и пароль», входными данными для которого являются пользовательские данные и «Проверка данных на валидность», в качестве управления для которого служат «Требования валидации БД»

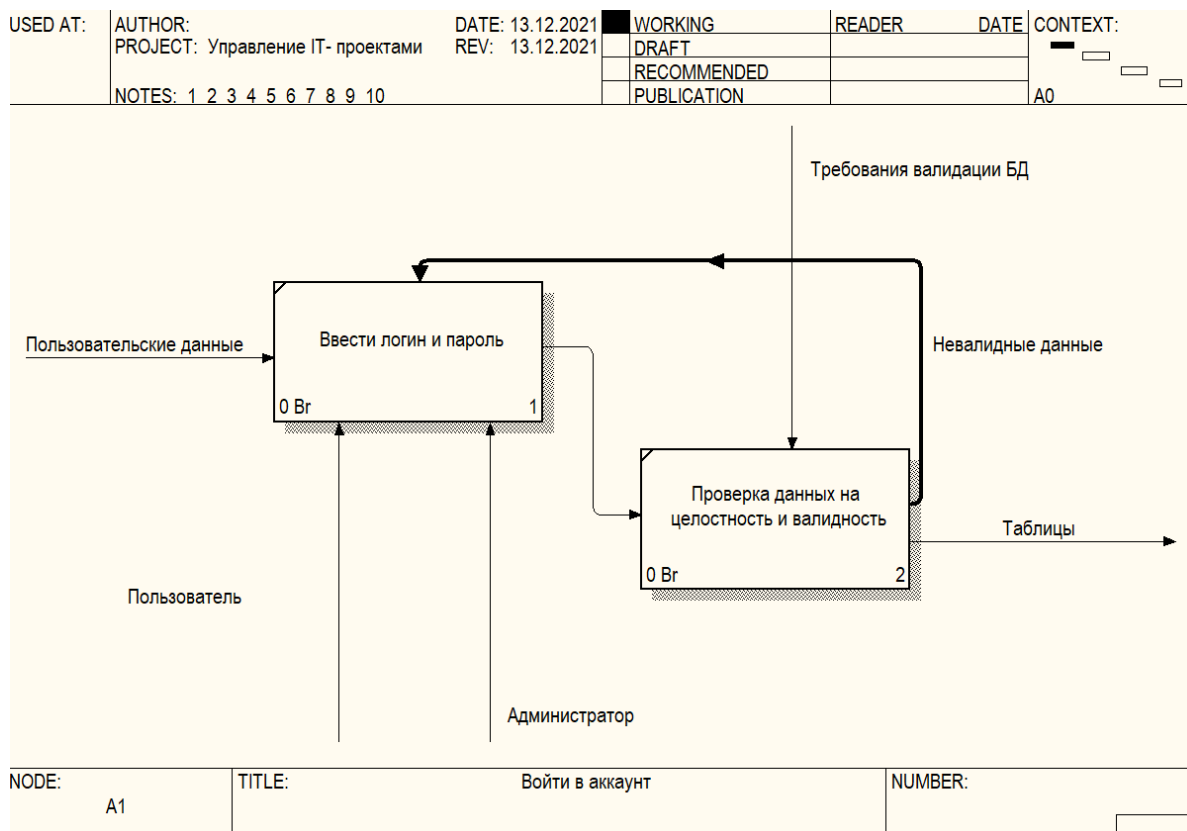


Рисунок 1.3 – Декомпозиция функционального блока «Войти в аккаунт»

Переходим к декомпозиции следующего функционального блока «Выбрать таблицу» (см. рисунок 1.4).

В данном проекте представлены три таблицы: ИТ-проектов, пользователей и таблица отчётности. Пользователь имеет доступ к таблице ИТ-проектов и отчётности, в то время как администратор – к ИТ-проектам и таблице пользователей.

Администратор может добавлять/редактировать/удалять данные из таблицы пользователей и таблицы ИТ-проектов, тем самым вносить изменения в БД. Пользователь может добавлять/редактировать/удалять данные из таблицы отчётности и таблицы ИТ-проектов, тем самым вносить изменения в БД.

Когда пользователь вносит изменения в таблицу отчётности, непосредственно меняя риски проектов, он изменяет аналитику.

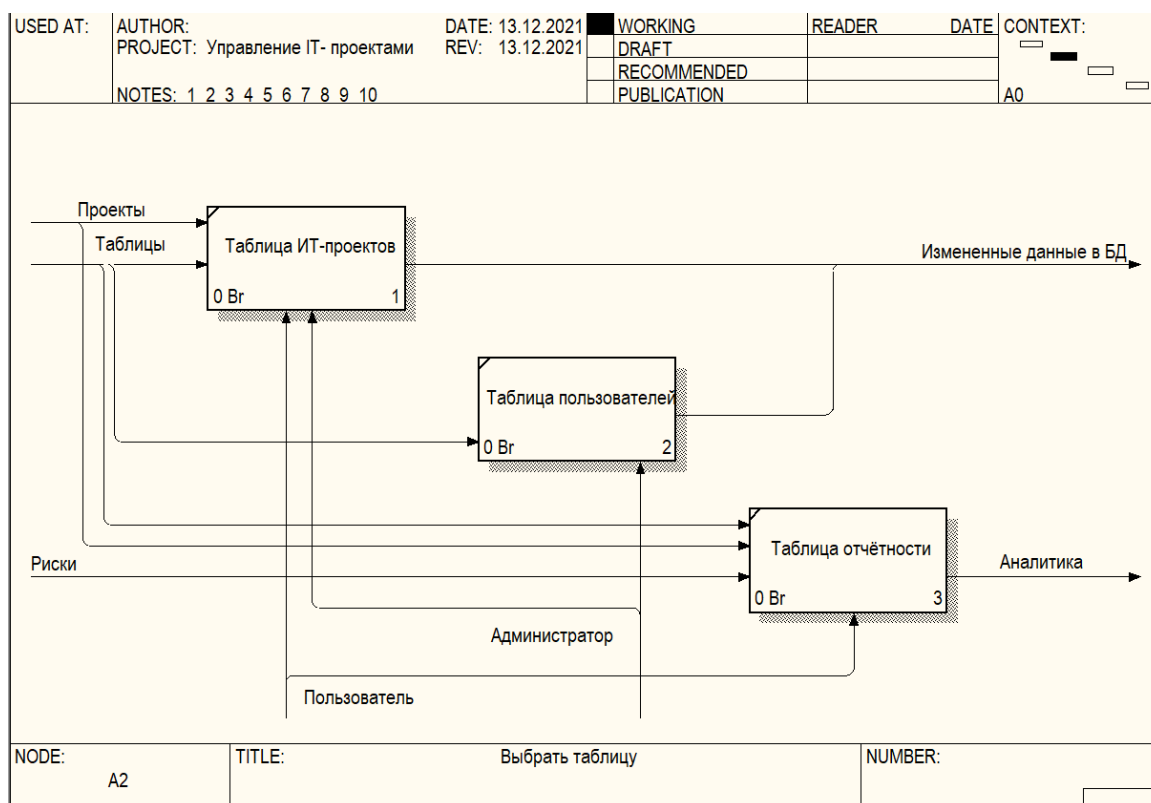


Рисунок 1.4 – Декомпозиция функционального блока «Выбрать таблицу»

Рассмотрим декомпозицию функционального блока «Просмотреть аналитику» (см. рисунок 1.5).

У администратора есть право работать с аналитикой уровня готовности проектов, по которой он может создать отчёт. Управляет этой аналитикой документация.

У пользователя есть право работать с анализом заказчиков, по которой он так же может создать отчётность.

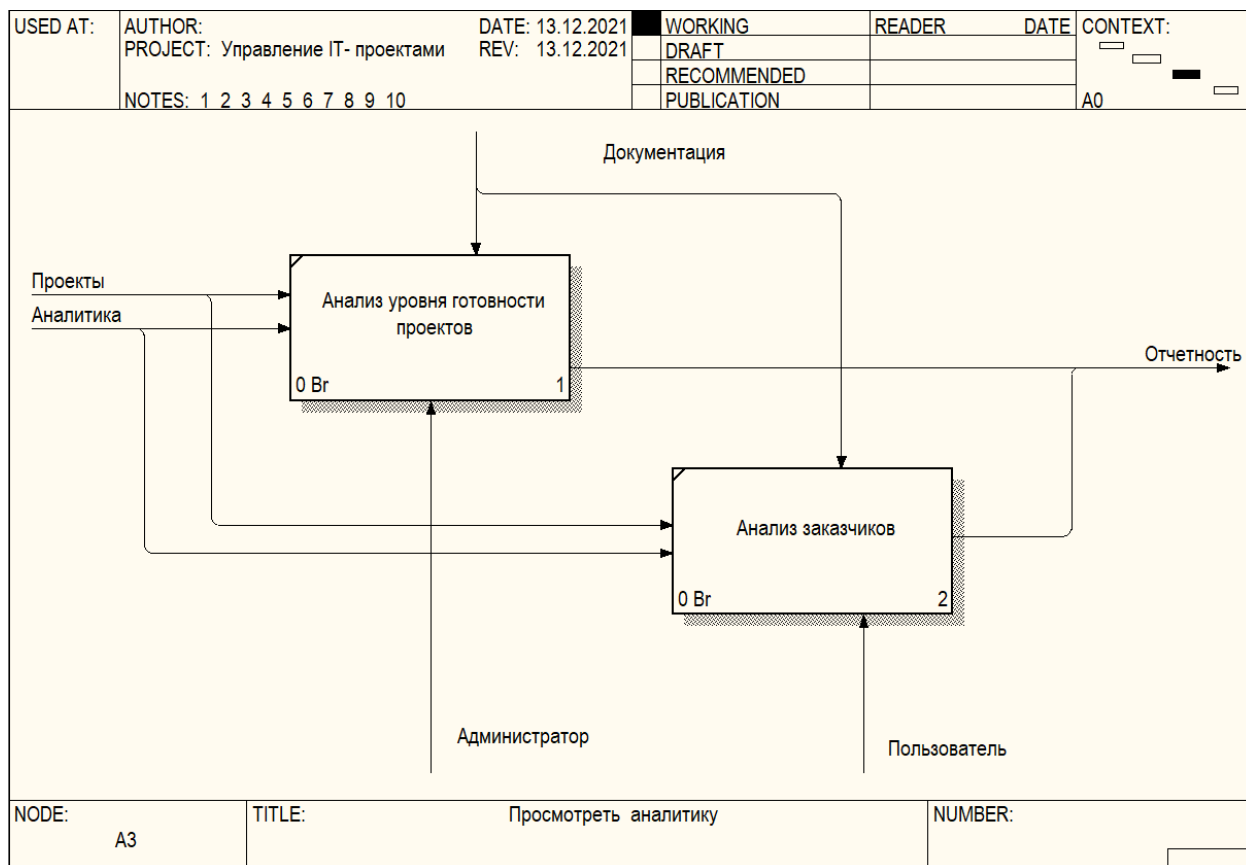


Рисунок 1.5 – Декомпозиция функционального блока «Просмотреть аналитику»

Целью построение функциональной модели является описание всех необходимых процессов с точностью, достаточной для однозначного моделирования деятельности системы.

Одним из положительных результатов построения функциональной модели оказывается прояснение границ моделирования системы в целом и ее основных компонентов. Хотя и предполагается, что в процессе работы над моделью будет происходить некоторое изменение границ моделирования, их вербальное (словесное) описание должно поддерживаться с самого начала для обеспечения координации работы участвующих в проекте аналитиков. Как и при определении цели моделирования, отсутствие границ затрудняет оценку степени завершенности модели, поскольку границы моделирования имеют тенденцию к расширению с ростом размеров модели [3].

4 ИНФОРМАЦИОННАЯ МОДЕЛЬ СИСТЕМЫ И ЕЕ ОПИСАНИЕ

Информационная система – это коммуникационная и вычислительная система по сбору, хранению, обработке и передаче информации, снабжающая работников различного ранга той информацией, которая необходима им для реализации функций управления.

Так же информационная модель системы отражает информацию о предметной области, так называемой части реального мира, данные о которой должны храниться в проектируемой базе данных. В данном случае предметной областью является аэропорт. При этом информация, необходимая для описания информационной системы, зависит от типа самой инфраструктуры.

Для того, чтобы обеспечить минимальную избыточность данных и физического объёма данных, а также ускоренный доступ, необходимо привести информационную модель к нормальной форме. При этом система называется нормализованной только в том случае, если она способна удовлетворять следующим требованиям: надежное хранение и обновление данных. Это поможет сократить риск потери данных или их искажения при внесении в БД. [6]

Для моделирования системы используются три этапа проектирования:

- концептуальное проектирование;
- логическое проектирование;
- физическое проектирование;

Концептуальное (инфологическое) проектирование – анализ предметной области и ее описание. Этот этап осуществляется без ориентации на какие-либо конкретные программные или технические средства [5].

Даталогическое (логическое) проектирование – описание логической структуры данных средствами системы управления базами данных (СУБД), для которой проектируется БД. Описание данной модели основывается на построении концептуальной модели. Даталогическое проектирование включает в себя:

- описание таблиц;
- описание связей между таблицами;
- описание атрибутов.

В данной информационной модели содержатся различные сущности. Под сущностями следует понимать, что это объект предметной области, который исследуется и моделируется. Иными словами, сущность – это любой различимый объект, о котором необходимо хранить информацию в базе данных.

Рассмотрим информационную модель предметной области. В работе было выделено пять сущностей (см. рисунок 2.1):

- admin;
- users;
- projects;
- customers;
- otchet.

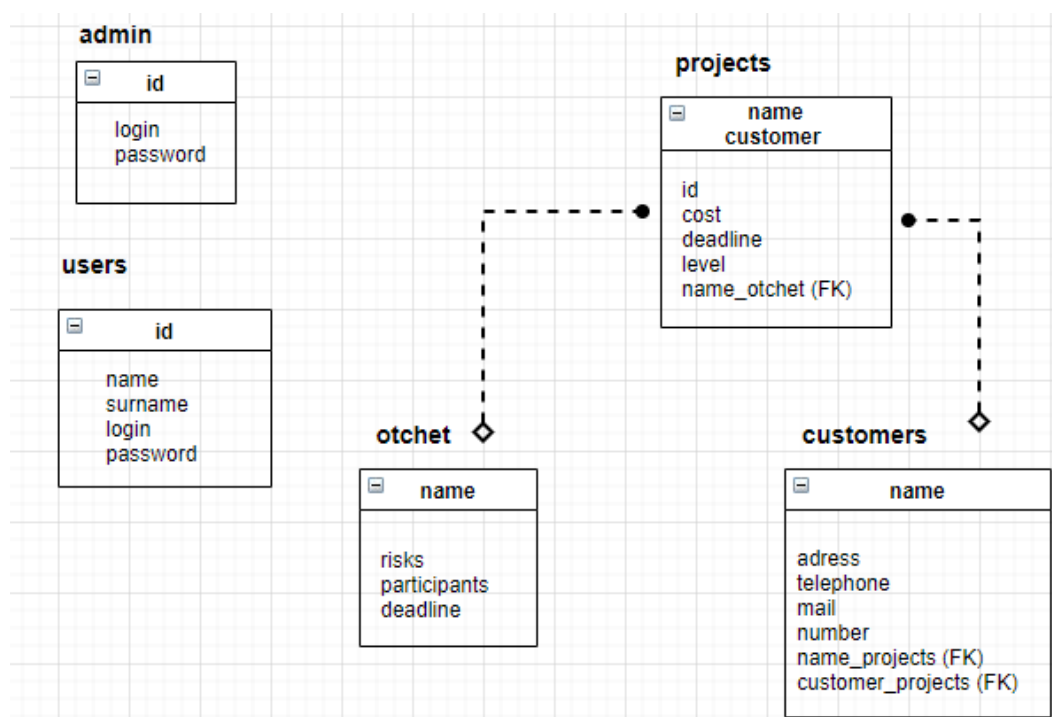


Рисунок 2.1 – Логическая модель системы

Физическое проектирование – описание физической структуры БД, т.е. ее размещения на запоминающем устройстве (рисунок 4.2):

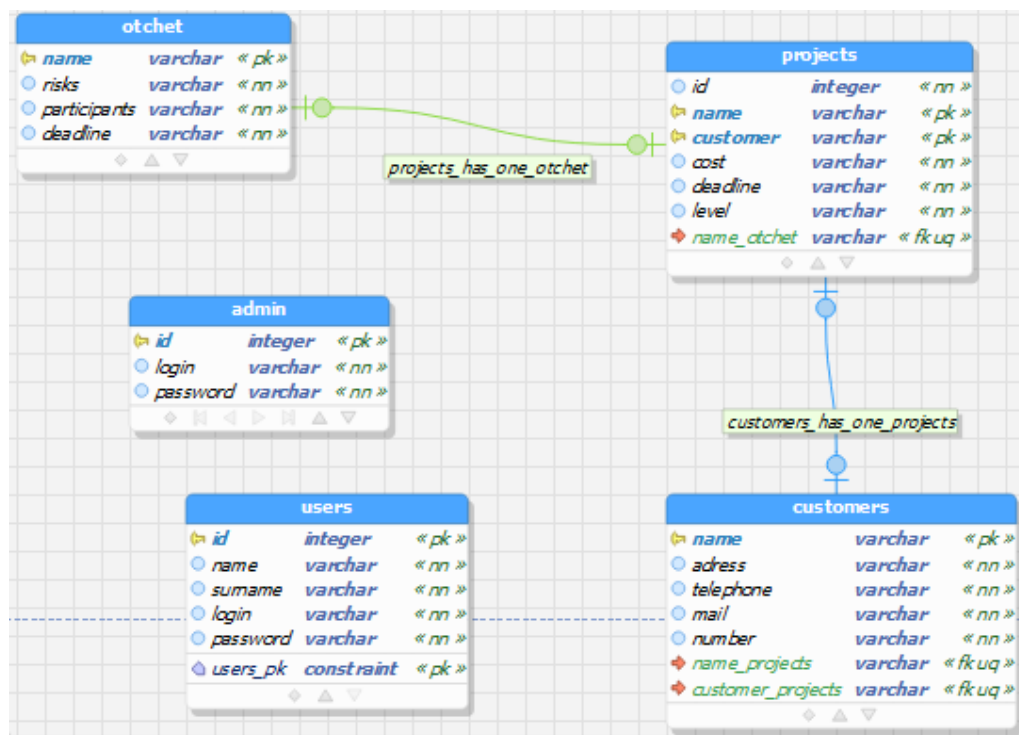


Рисунок 2.2 – Физическая модель системы

Сущность `users` представляет данные о зарегистрированных пользователях и описывается следующими атрибутами:

- `id` – идентификатор пользователя, для каждого аккаунта;
- `name` – имя пользователя;
- `surname` – фамилия пользователя;
- `password` – хранит пароль, по которому пользователь может пройти авторизацию (хранится в зашифрованном виде);

Сущность `admin` предназначена для хранения данных администраторов и описывается следующими атрибутами:

- `id` – идентификатор администратора;
- `login` – логин аккаунта, по которому администратор может пройти авторизацию;
- `password` – идентификатор аккаунта, которому принадлежит администратор.

Сущность `otchet` предназначена для хранения возможных рисков и создания отчётности и описывается следующими атрибутами:

- `risks` – возможные риски ИТ-проектов;
- `name` – название ИТ-проекта;
- `participants` – участники ИТ-проекта;
- `deadline` – сроки сдачи ИТ-проекта.

Сущность projects предназначена описания ИТ-проектов и описывается следующими атрибутами:

- id – идентификатор ИТ-проекта;
- name – название ИТ-проекта;
- customer – заказчик ИТ-проекта;
- cost – стоимость ИТ-проекта;
- deadline – срок сдачи ИТ-проекта;
- level – уровень выполненности/готовности ИТ-проекта.

Сущность projects и otchet связаны отношением “один-к-одному”, т.е. к одному проекту может быть привязан один риск.

Сущность customers предназначена для хранения данных заказчиков ИТ-проектов и описывается следующими атрибутами:

- adress – фактический адрес заказчиков;
- name – заказчики;
- telephone – телефон для связи с заказчиками;
- mail – почта для связи с заказчиками;
- number – количество ИТ-проектов, которые заказали заказчики.

Сущность projects и customers связаны отношением “один-к-одному”, т.е. к одному проекту может быть привязан один заказчик.

5 ОПИСАНИЕ АЛГОРИТМА, РЕАЛИЗУЮЩЕГО БИЗНЕС-ЛОГИКУ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ

Приложение разработано на основе архитектуры «клиент-сервер». Для создания насыщенного графического интерфейса была использована библиотека JavaFX. Работа с информационной моделью системы была осуществлена с помощью объектно-реляционной системой управления базами данных – PostgreSQL.

Для начала работы необходимо осуществить идентификацию для базы данных. Пользователь вводит данные на стороне сервера. При этом необходимо подключить клиента по уникальному порту. После обработки данных в консоли сервера выводится сообщения об успешном установлении соединения. Затем сервер отправляет сообщение клиенту о результатах выполнения своей работы. Выведенное сообщение отображает результат выполненных работ.

Далее пользователю необходимо авторизоваться/зарегистрироваться. Для аторизации в соответствующей форме необходимо ввести уникальные пароль и логин. После авторизации данные передаются на сервер. Для корректной идентификации пользователя был разработан метод «getUserAuthorization» класса «User». Данные обрабатываются на стороне базы данных, где хранится информация о зарегистрированных пользователях. В случае успешной авторизации выводится сообщение о выполненной работе. Также пользователь может зарегистрироваться в системе. Для этого используется метод «getUserRegistration» класса «User». Код данных функций представлен ниже.

```
public ResultSet getUserAuthorization(User user) { //проверка на
    существование пользователя

    ResultSet resSet = null;
    String select = "SELECT " + Constants.LOGIN + "," +
    Constants.PASSWORD + " FROM " + Constants.USERS TABLE +
        " WHERE " + Constants.LOGIN + "'=? " + "AND " +
    Constants.PASSWORD + " =? ";
    try {
        PreparedStatement preparedStmt =
    connection.prepareStatement(select);
        preparedStmt.setString(1, user.getLogin());
        preparedStmt.setString(2, user.getPassword());

        resSet = preparedStmt.executeQuery();
        System.out.println("Such user exists !");
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
    return resSet;
}

public void getUserRegistration(User user) { //регистрация пользователя
    try {
```

```

        String select = "insert into " + Constants.USERS_TABLE +
"(login, password, name, surname) "
        + "values (?, ?, ?, ?)";
        PreparedStatement preparedStmt =
connection.prepareStatement(select);
        preparedStmt.setString(1, user.getLogin());
        preparedStmt.setString(2, user.getPassword());
        preparedStmt.setString(3, user.getName());
        preparedStmt.setString(4, user.getSurname());
        preparedStmt.execute();
        System.out.println("User was registred !");
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

```

Алгоритм авторизации и регистрации пользователя представлен на рисунках 3.1 и 3.2 соответственно.

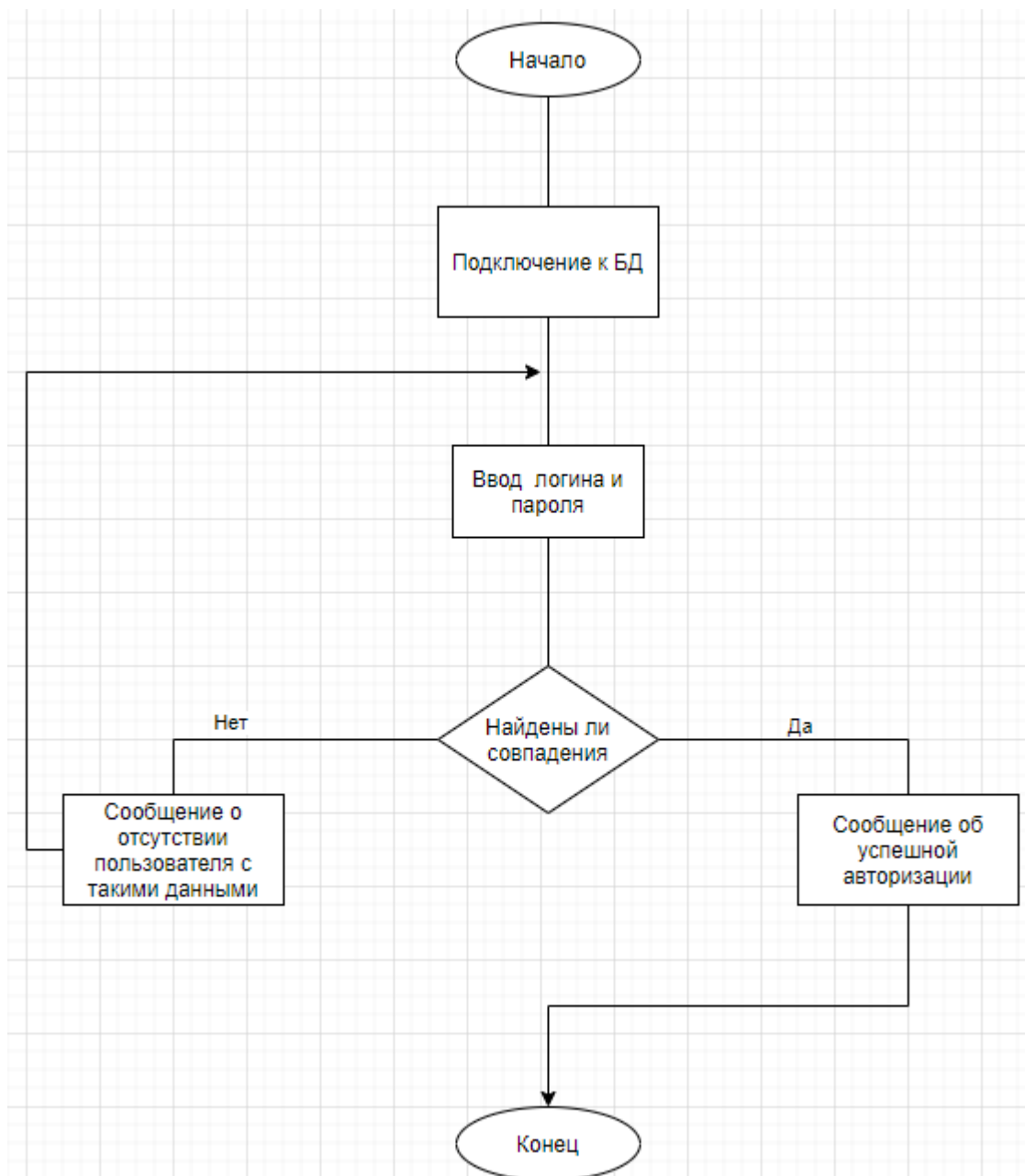


Рисунок 3.1 – Блок-схема алгоритма авторизации пользователя

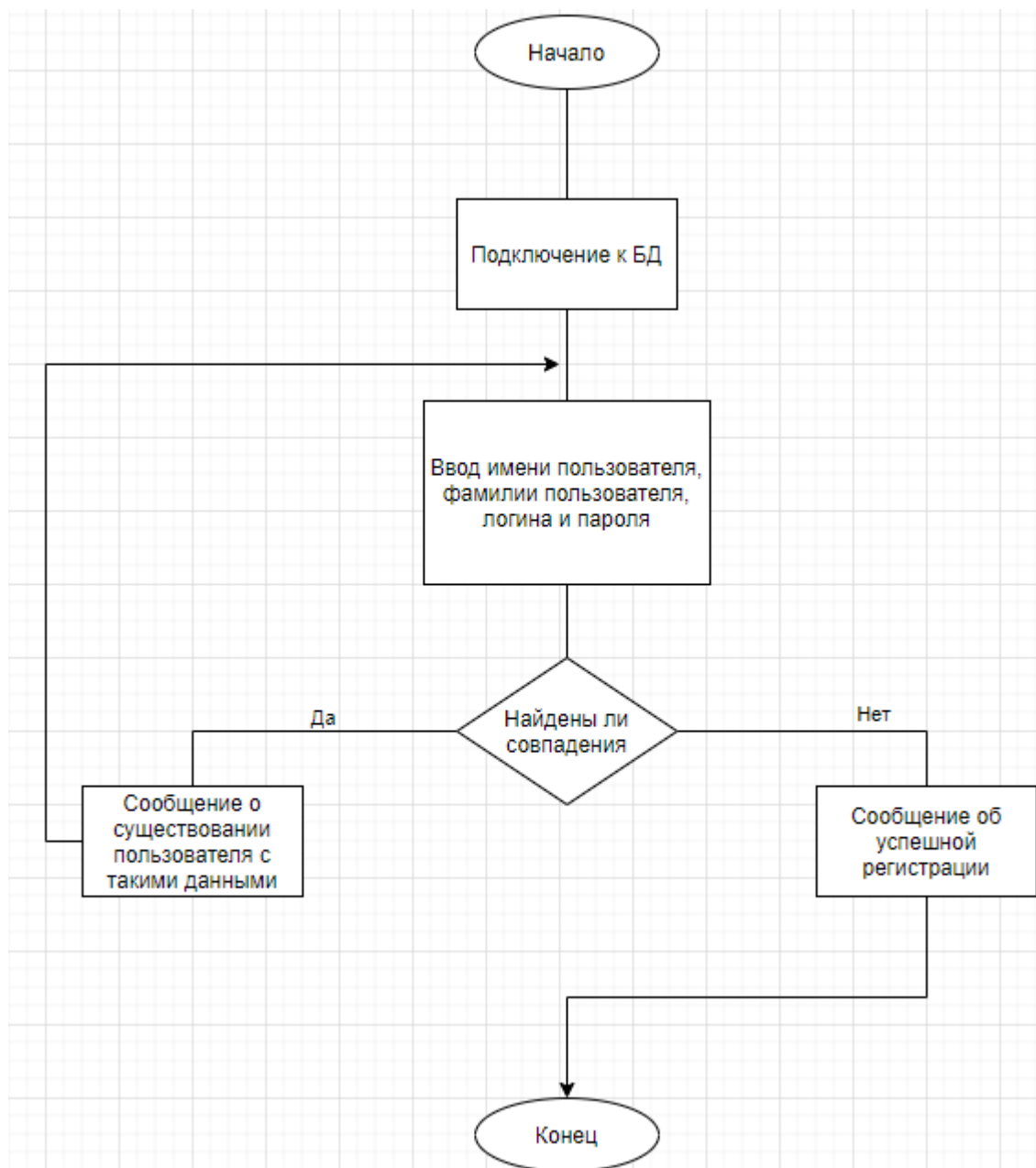


Рисунок 3.2 – Блок-схема алгоритма регистрации пользователя

После авторизации/регистрации пользователю представлены некоторые таблицы для работы. Основной из алгоритмов работы с таблицами является алгоритм работы функции редактирования. При нажатии на строку с нужным ИТ-проектом, значения столбцов помещаются в поля для редактирования внизу окна. Далее пользователь может выборочно отредактировать информацию и нажать на кнопку «Редактировать». После нажатия на кнопку, данные передаются на сервер. На сервере вызывается метод, соответствующий запросу редактирования, и обновляет данные в таблице. Результат обновления передается пользователю.

Алгоритм работы функции редактирования представлен на рисунке 3.3.

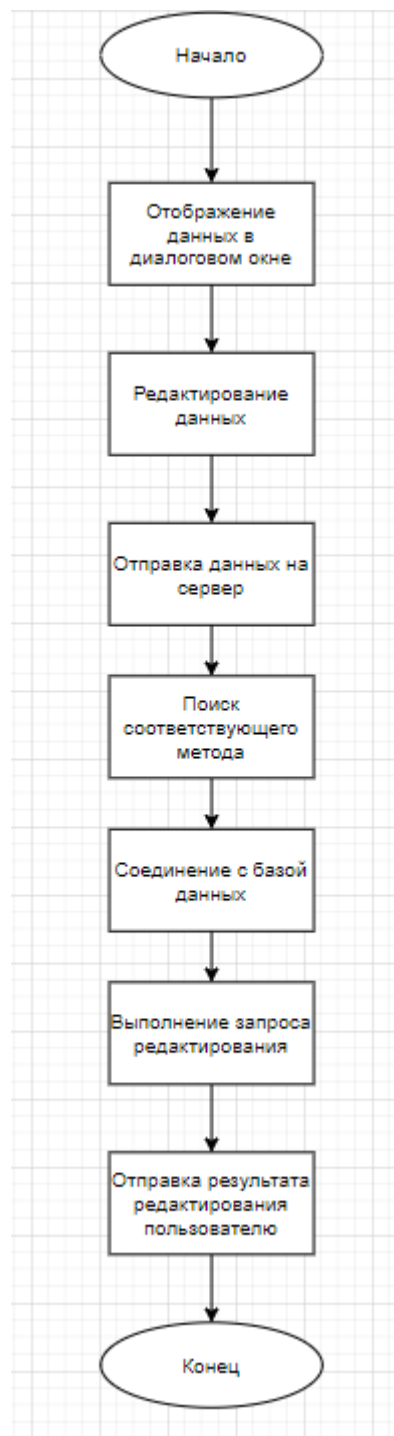


Рисунок 3.3 – Блок-схема алгоритма работы функции редактирования

Бизнес-логика серверной части приложения осуществляет обработку всех запросов пользователя. Это реализуется как со стороны администратора, так и со стороны пользователя. Также на сервере осуществляется обработка исключительных ситуаций, которые могут произойти в ходе работы.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректного запуска системы, компьютер или рабочая станция, на которой разворачиваются клиентская и серверная части приложения должен обладать следующими минимальными требованиями

- 32х разрядная система ОС Windows 7 и выше;
- СУБД PostgreSQL;
- Java Development Environment (IntelliJ IDEA Ultimate).

После успешного запуска серверной части приложения, а потом клиентской, появляется окно, которое необходимо для выбора роли (рисунок 4.1).



Рисунок 4.1 – Выбор роли

После выбора роли Администратор следует окно авторизации (рисунок 4.2).

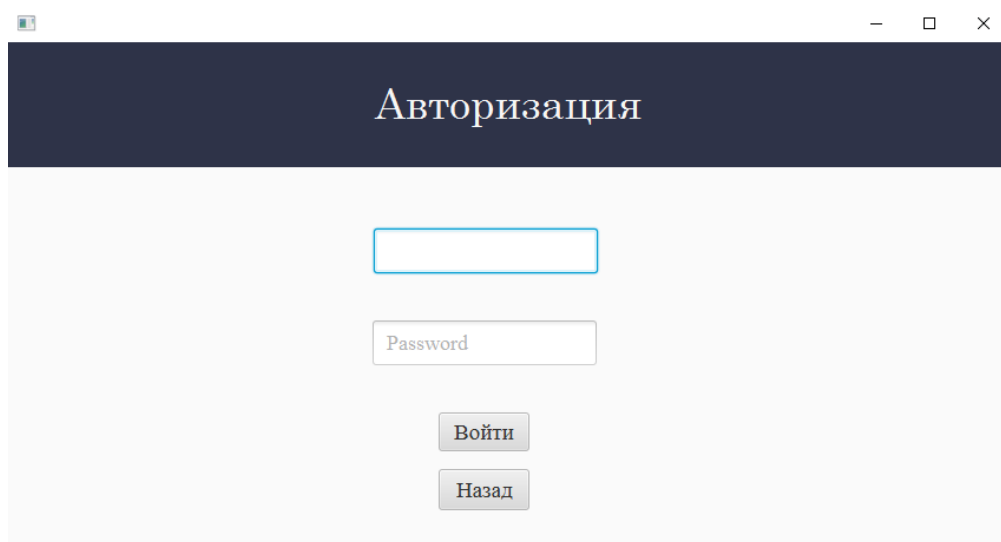


Рисунок 4.2 – Окно авторизации администратора

Необходимо ввести валидные данные в поля Логин и Пароль. После этого Администратору будет доступен весь его функционал (рисунок 4.3).

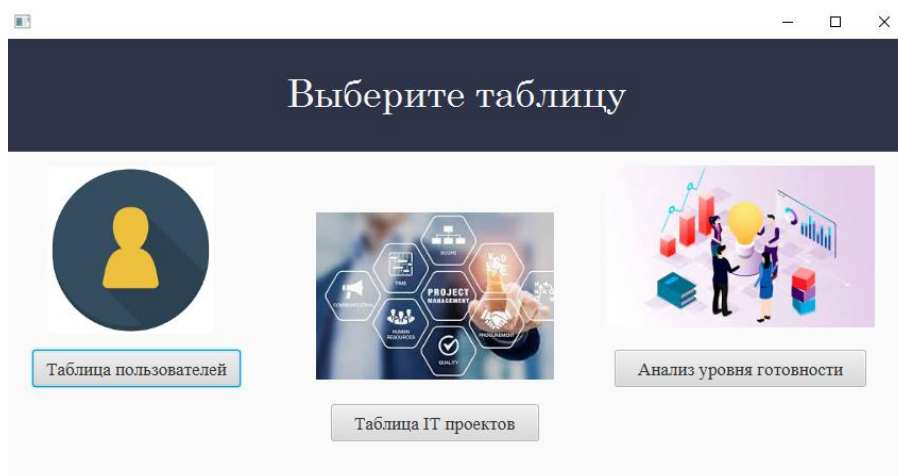


Рисунок 4.3 – Главное окно администратора

У администратора есть доступ к трем пунктам: «Таблица пользователей», «Таблица ИТ-проектов» и «Анализ уровня готовности проектов».

Перейдём к первой таблице. В данном окне реализован следующий функционал: просмотр данных из БД, удаление пользователей из БД, поиск по имени и фамилии пользователя. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.4).

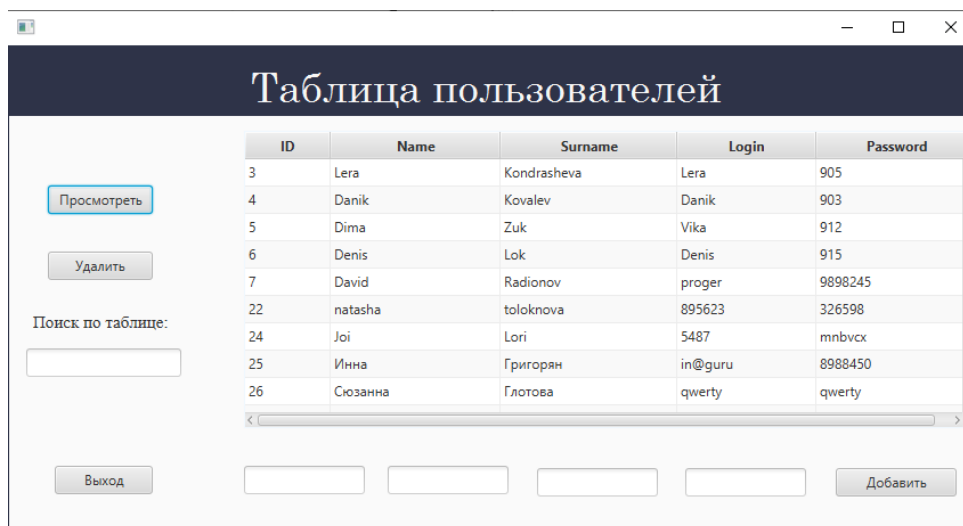


Рисунок 4.4 – Таблица пользователей

Следующая таблица, к которой есть доступ у Администратора – таблица ИТ-проектов. В данной таблице предусмотрен следующий функционал: просмотр ИТ-проектов из БД, добавление ИТ-проектов в БД, поиск по названию ИТ-проекта и заказчику, редактирование и добавление данных в БД. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.5).



Рисунок 4.5 – Таблица ИТ-проектов

Рассмотрим третью таблицу. На ней предстала круговая диаграмма, которая иллюстрирует процент готовности ИТ-проектов компании. По данной диаграмме можно сформировать отчет с дальнейшим его использованием вне данной системы. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.6).



Рисунок 4.6 – Анализ уровня готовности ИТ-проектов

После выбора роли Пользователь следует окно авторизации, а также при необходимости регистрация (рисунок 4.7 и 4.8).

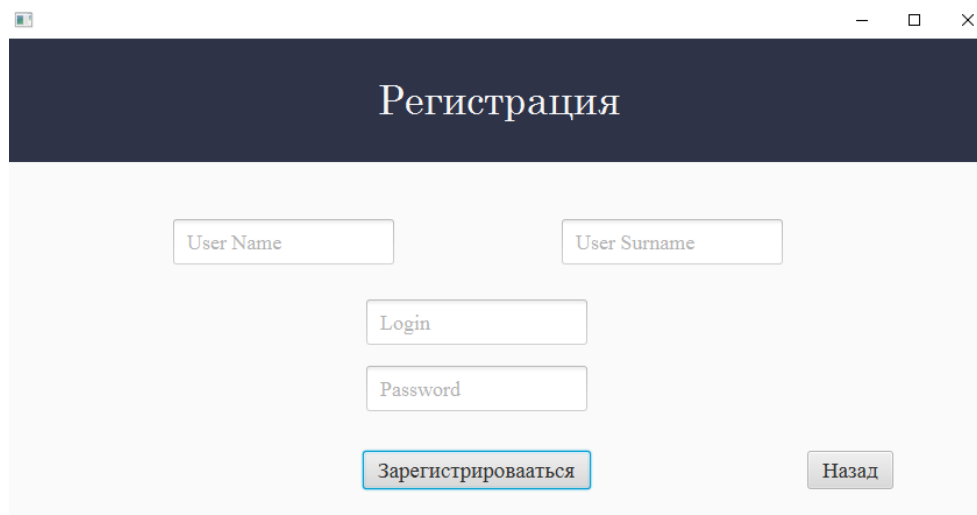
Авторизация

Password

Войти

Зарегистрироваться

Рисунок 4.7 – Авторизация Пользователя



Регистрация

User Name

User Surname

Login

Password

Зарегистрироваться

Назад

Рисунок 4.8 – Регистрация Пользователя

После авторизации/регистрации у пользователя есть доступ к трем пунктам: «Анализ заказчиков», «Таблица ИТ-проектов» и «Составление отчётности» (рисунок 4.9).

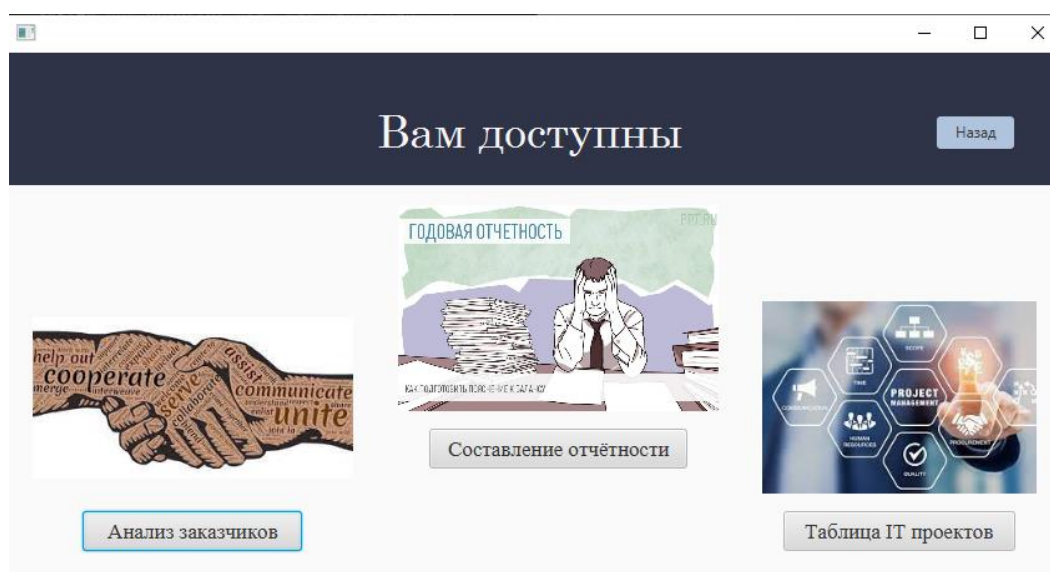


Рисунок 4.9 – Главное окно пользователя

Перейдём к первой таблице. На ней предстала круговая диаграмма, которая иллюстрирует процент заказчиков ИТ-проектов компании. По данной диаграмме можно сформировать отчёт с дальнейшим его использованием вне данной системы. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.10).

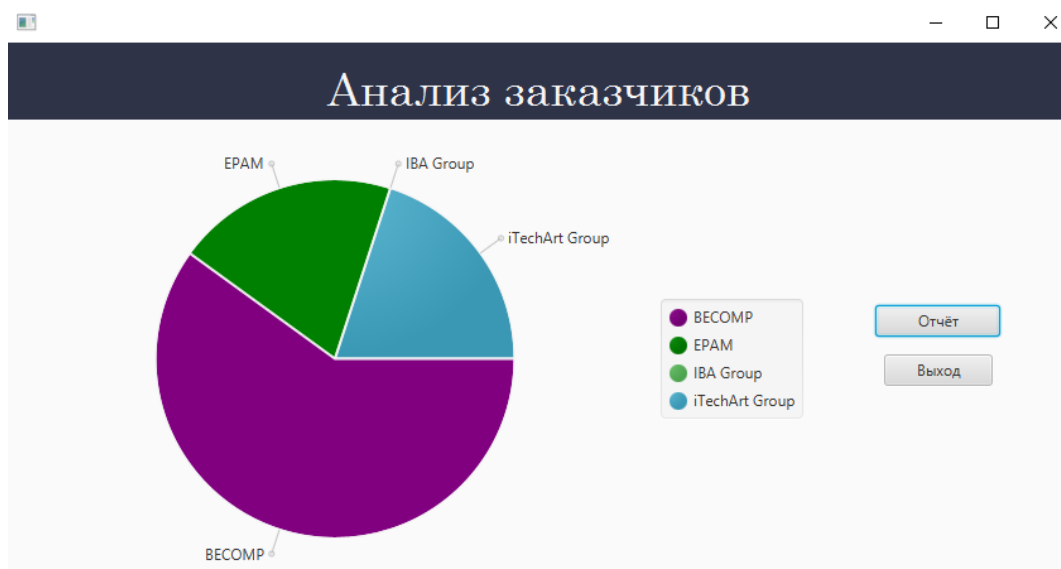


Рисунок 4.10 – Анализ заказчиков

Следующая таблица, к которой есть доступ у Пользователя – таблица отчётности. В данной таблице предусмотрен следующий функционал: просмотр ИТ-проектов, возможных рисков, участников проекта и сроки выполнения данного ИТ-проекта. По данной таблице можно сформировать отчёт с дальнейшим его использованием вне данной системы. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.11).

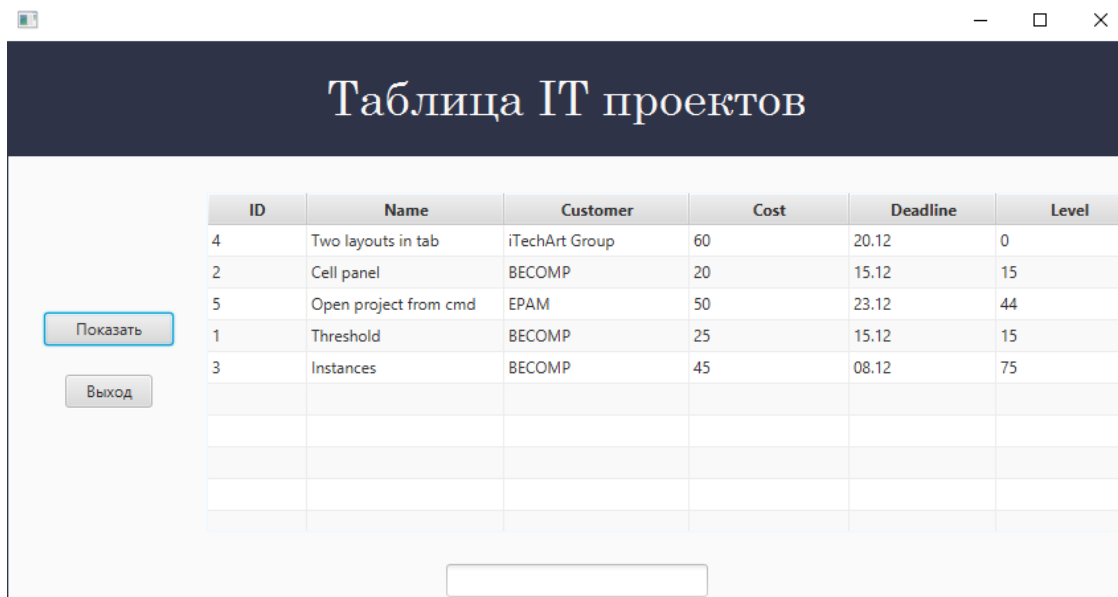
Таблица отчётности

Name	Risks	Participants	Deadline
Threshold	Not displayed on layout	Programmers, testers	12.12
Cell panel	Endless scroll	Programmers, testers	15.12

Рисунок 4.11 – Таблица отчётности

Третья таблица – таблица ИТ-проектов. В данной таблице предусмотрен следующий функционал: просмотр ИТ-проектов из БД и поиск по названию

ИТ-проекта, заказчику и стоимости ИТ-проекта. После окончания работы с данной таблицей для того, чтобы выйти, предусмотрена кнопка «Выход» (рисунок 4.12).



ID	Name	Customer	Cost	Deadline	Level
4	Two layouts in tab	iTechArt Group	60	20.12	0
2	Cell panel	BECOMP	20	15.12	15
5	Open project from cmd	EPAM	50	23.12	44
1	Threshold	BECOMP	25	15.12	15
3	Instances	BECOMP	45	08.12	75

Рисунок 4.12 – Таблица ИТ-проектов

Если при работе с приложением на серверной и клиентской части возникли трудности и формы отображаются не так, как на рисунках, то необходимо переустановить библиотеку JavaFX.

1 АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ

7.1 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.

Рассмотрим диаграмму вариантов использования для всего проекта.

Начальным состоянием данной диаграммы является открытие главного меню. Далее осуществляется выбор роли. Если это Администратор, то он авторизуется и имеет доступ к трем таблицам. Если это Пользователь, то он может как авторизоваться, так и зарегистрироваться. (см. рисунок 5.1).

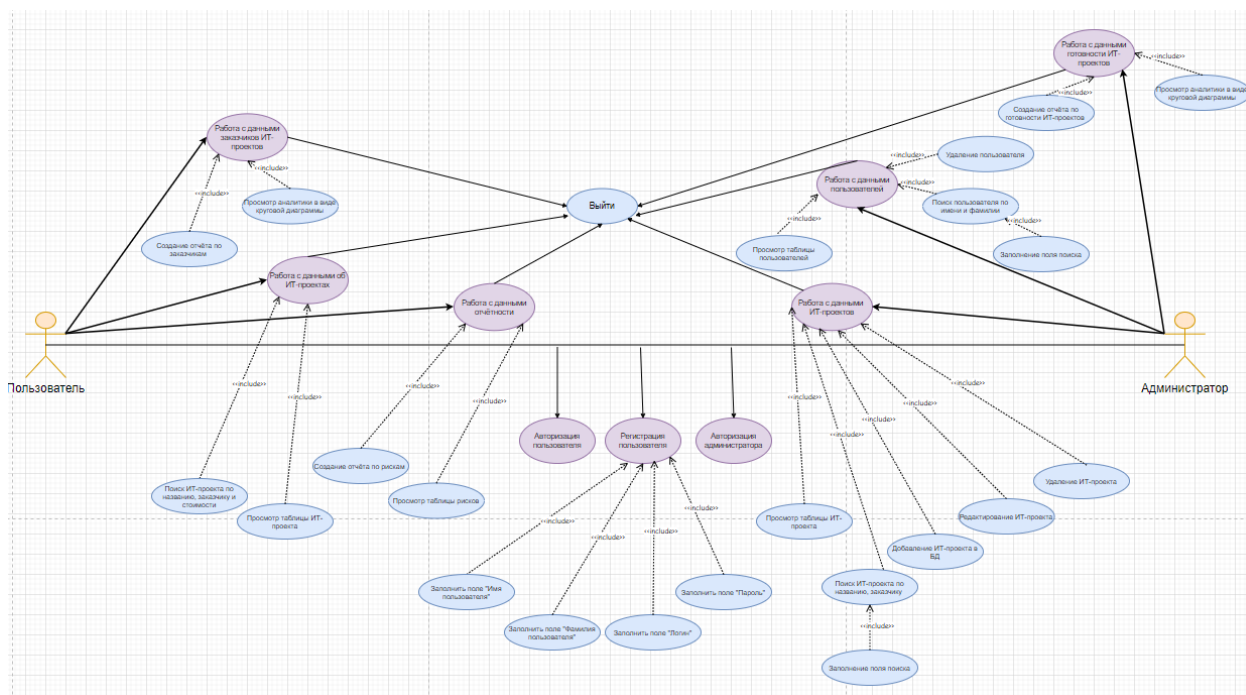


Рисунок 5.1 – Диаграмма вариантов использования

7.2 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 представлено на рисунке 5.2.

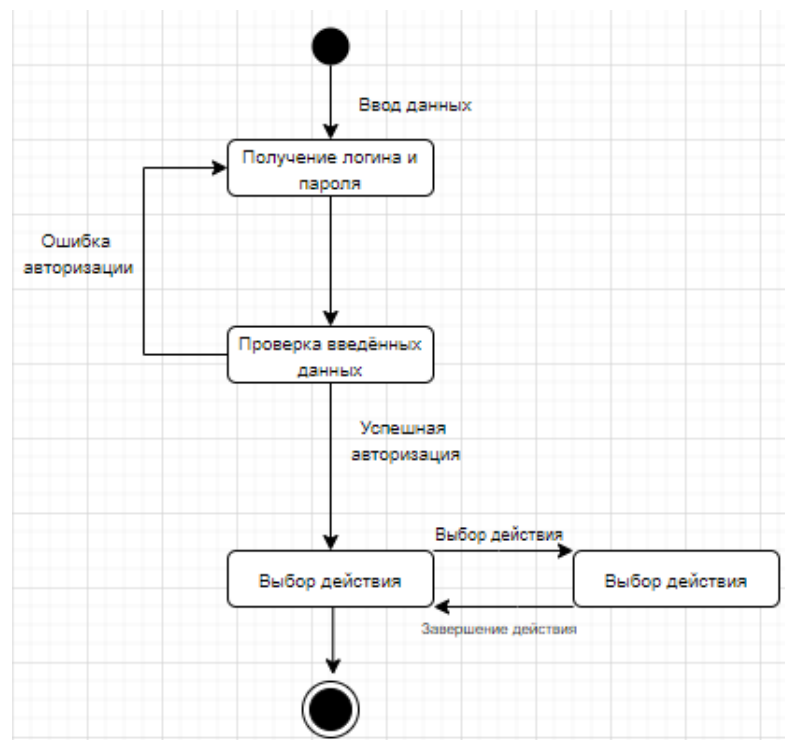


Рисунок 5.2 – Диаграмма состояний

7.3 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 представлено на рисунке 5.3.

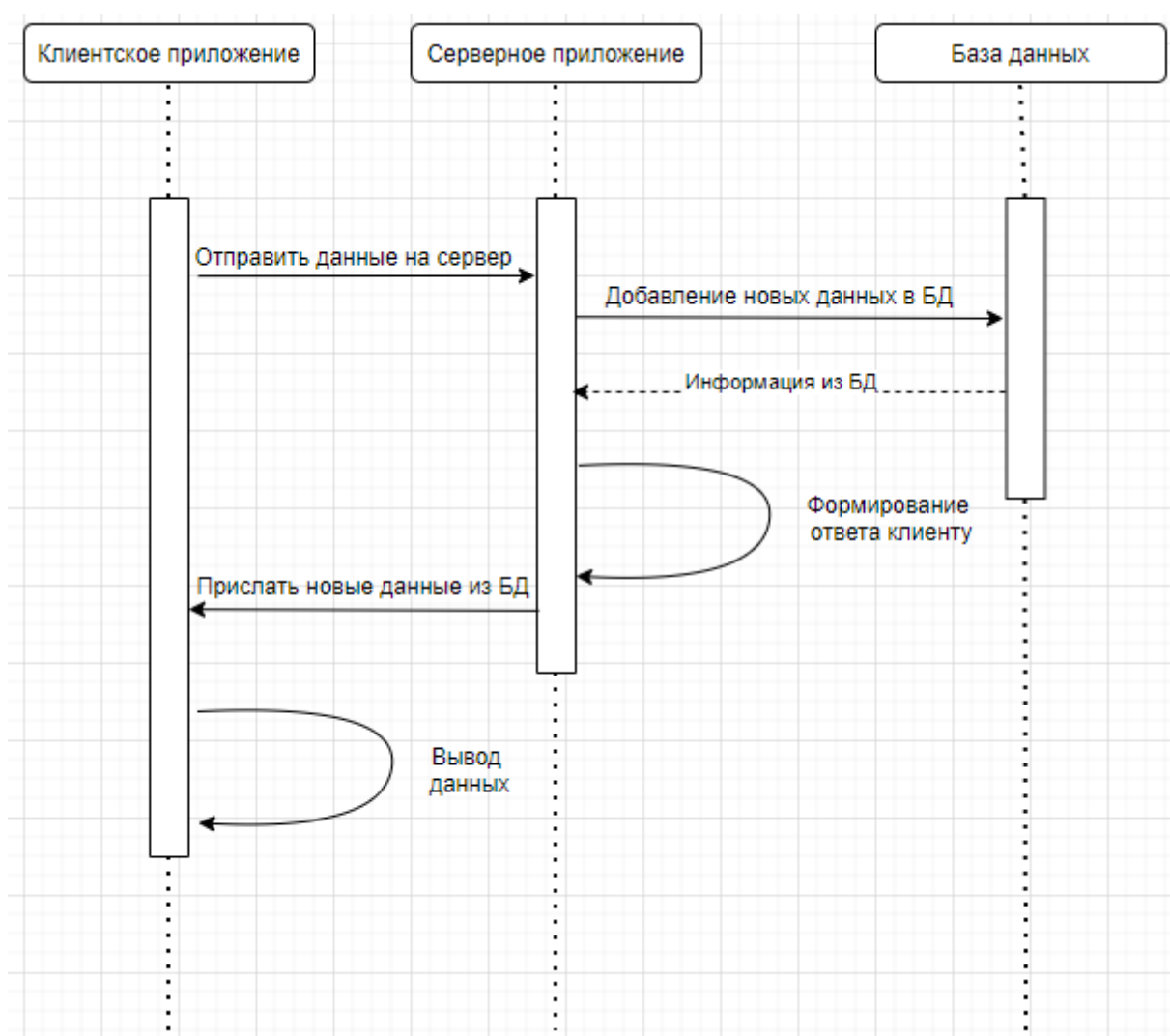


Рисунок 5.3 – Диаграмма последовательности

7.4 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 представлено на рисунке 5.4.

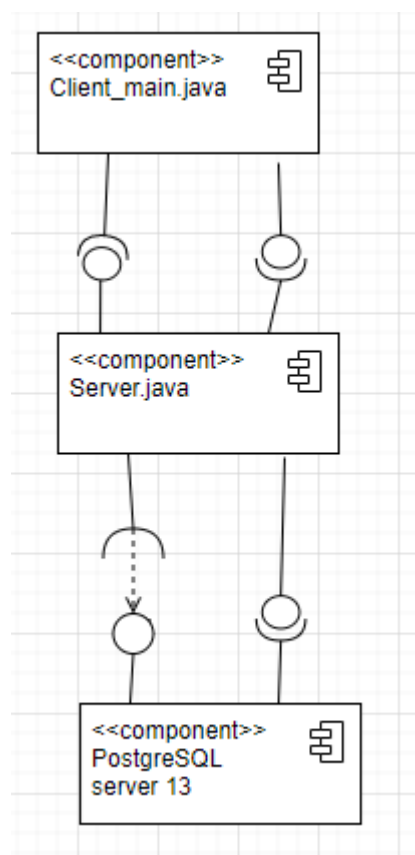


Рисунок 5.4 – Диаграмма компонентов

7.5 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 представлено на рисунке 5.5.

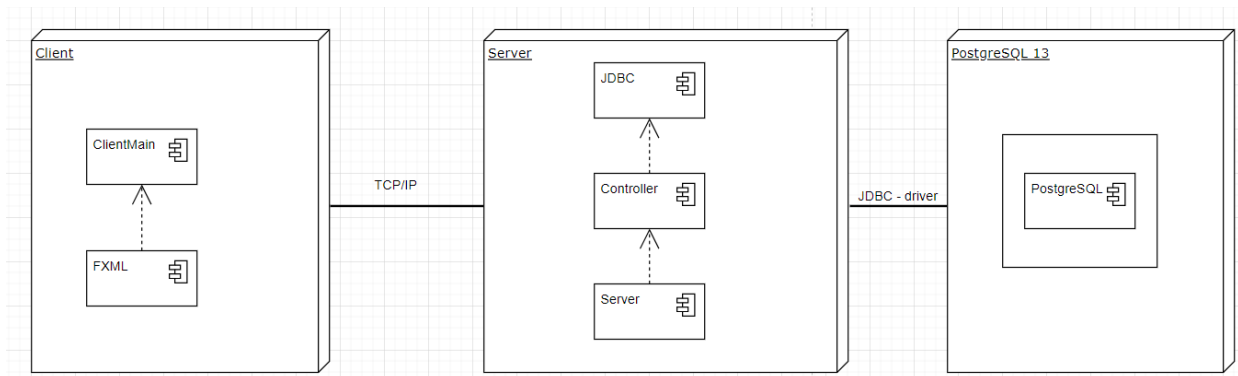


Рисунок 5.5 – Диаграмма развёртывания

7.6 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 представлено на рисунке 5.6–5.9.

Project	
m	Project()
m	Project(Integer, String, String, String, String, String)
m	Project(String, String)
m	Project(int, String, String, String, String, String)
m	toString() String
p	cost String
p	customer String
p	deadline String
p	id int
p	level String
p	name String

User	
m	User()
m	User(String, String, String, String)
m	User(int, String, String, String, String)
m	toString() String
p	id int
p	login String
p	name String
p	password String
p	surname String

Otchet	
m	Otchet()
m	Otchet(String, String, String, String)
m	toString() String
p	deadline String
p	name String
p	participants String
p	risks String

Admin	
m	Admin()
m	Admin(int, String, String)
m	toString() String
p	id int
p	login String
p	password String

Customer	
m	Customer(String, String)
m	toString() String
p	name String
p	number String

Рисунок 5.6 – Диаграмма классов пакета Model

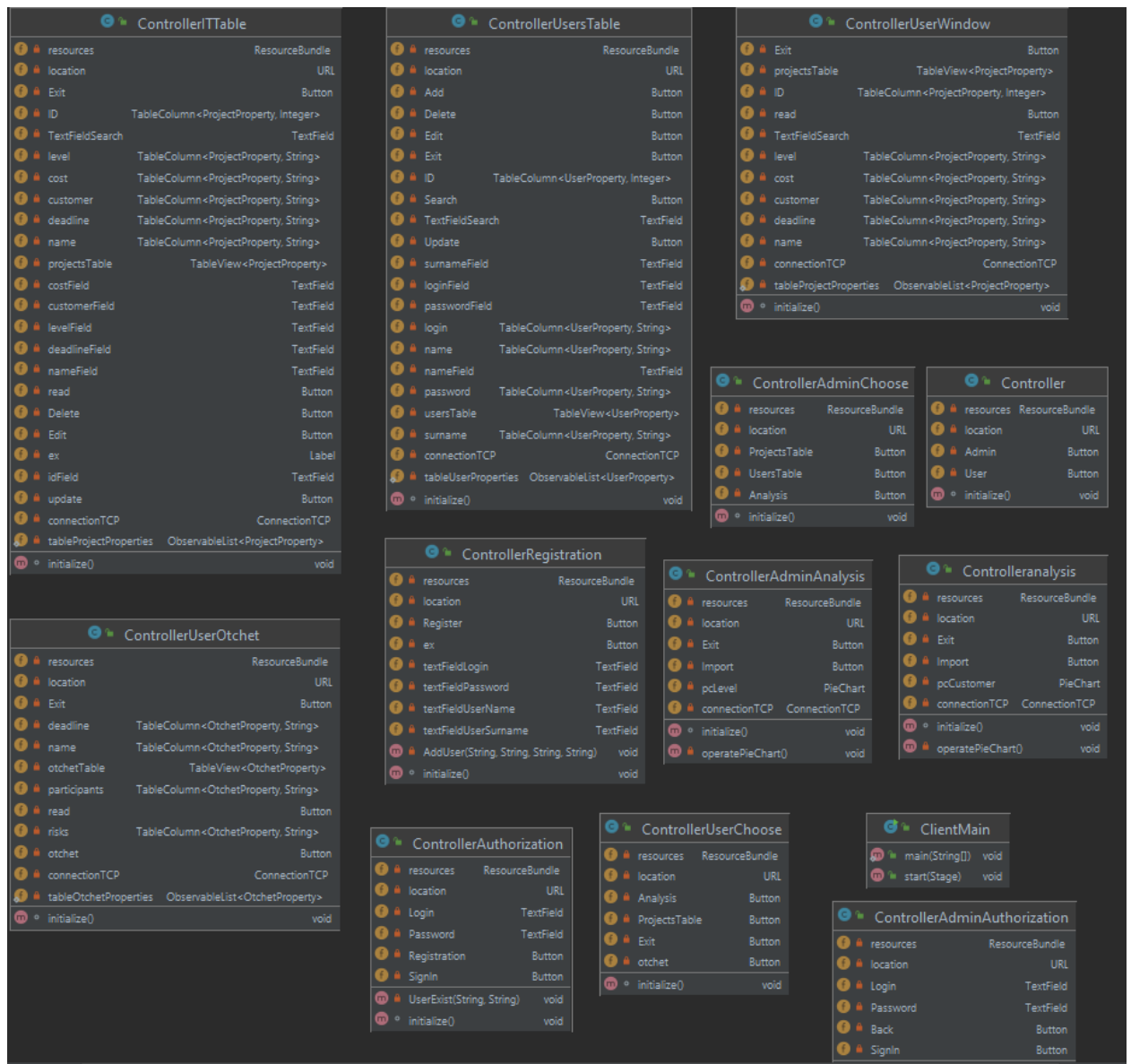


Рисунок 5.7 – Диаграмма классов пакета Controllers

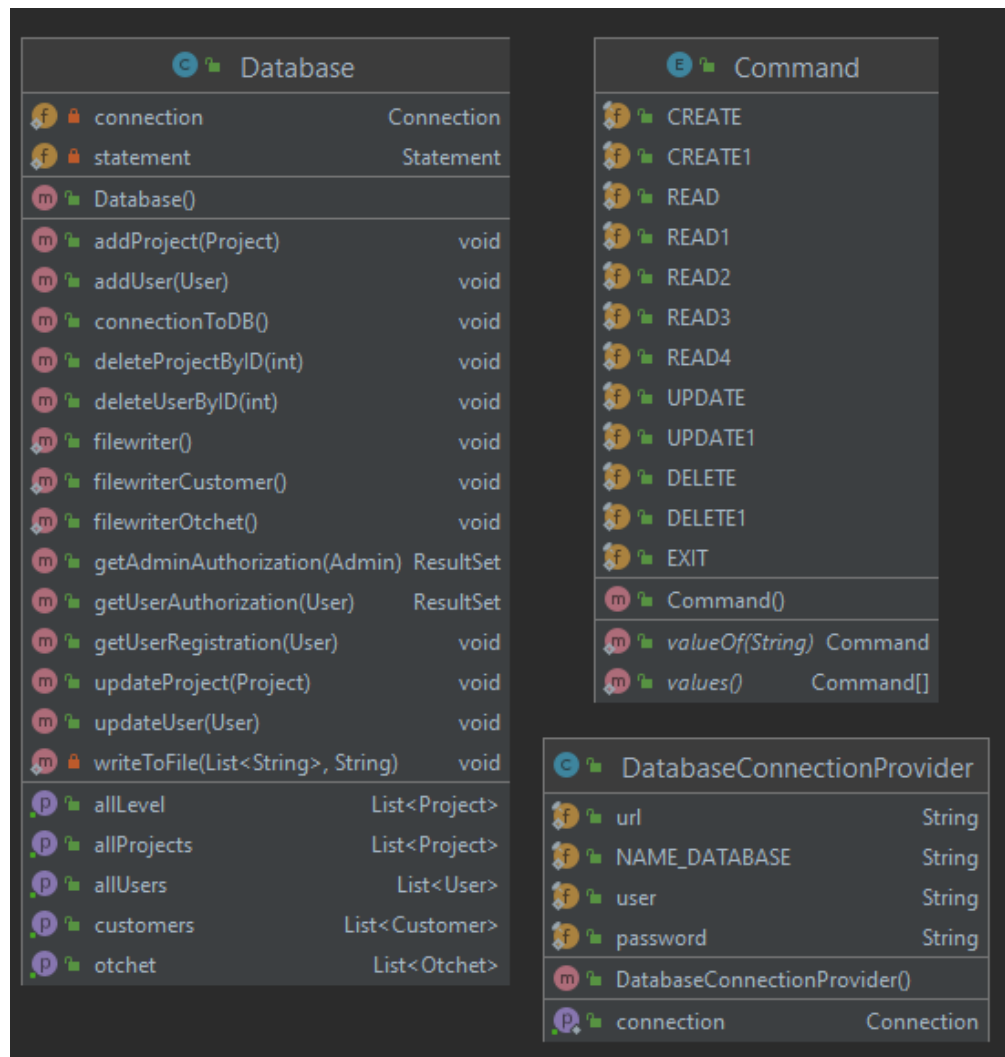


Рисунок 5.8 – Диаграмма классов пакета database

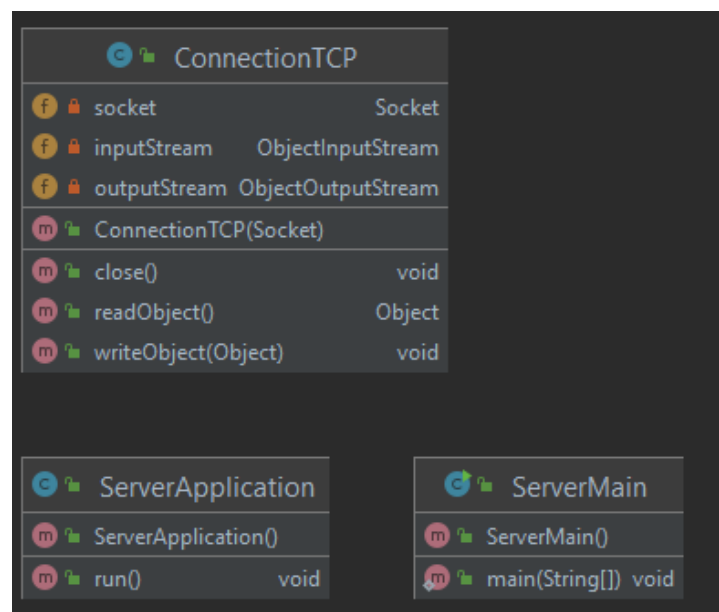


Рисунок 5.9 – Диаграмма классов пакета server

8 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ РАЗРАБОТАННОЙ СИСТЕМЫ

Один из важнейших этапов производства ПО направлен на детальное изучение кода программы и выявление багов в работе. Этот этап называется тестированием системы.

Тестирование ПО проводится, чтобы:

- конечный продукт соответствовал всем описанным требованиям;
- приложение в будущем работало правильно при любых обстоятельствах;
- предоставления актуальной информации о состоянии продукта на данный момент.

В результате тестирования моего курсового проекта были выявлены некоторые ситуации, которые затрудняли использование приложения.

При вводе некорректных значений при авторизации ячейки для ввода начинают трястись, была реализована анимация.

ЗАКЛЮЧЕНИЕ

Главной целью курсового проекта была разработка приложения для повышение качества работы ИТ компании. При решении поставленной задачи был выбран ряд соответствующих технологий и инструментов, которые позволили осуществить корректную работу данного приложения.

При функциональном моделировании было создано точное описание проектируемой системы, а также интерпретация полученного описания для определения оценочных знаний некоторых характеристик системы.

Информационная модель системы была создана для корректной работы с данными: хранение, обработка, удаление. При этом данная модель способна предоставлять доступ к информации, необходимой пользователю.

Проектирование UML-диаграмм позволило сформулировать основные требования к информационной системе, а также обеспечить объективность в выработке требований к проектированию системы. Данные модели описания требований к информационной системе преобразуются в систему моделей, описывающих концептуальный проект информационной системы. При этом сформировались модели архитектуры информационной системы, требования к программному обеспечению и информационному обеспечению.

В рамках курсовой работы была продумана логика действий, спектр возможностей и функционал администратора и пользователя.

При выполнении курсовой работы были рассмотрены основные требования и задачи проектированной системы. Автоматизация управления ИТ-проектами была выполнена в полной мере, что позволяет увеличить эффективность и конкурентоспособность системы. Автоматизация была проведена в основных процессах разработки продукта. Благодаря этому появилась возможность обеспечить комфортное и своевременное взаимодействие с пользователем внутри компании.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Управление ИТ-проектом [Электронный ресурс]. – Режим доступа: <https://roi4cio.com/categories/category/upravlenie-it-proektom/>
- [2] Менеджмент ИТ-проекта [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/169693/>
- [3] Построение контекстной диаграммы [Электронный ресурс]. – Режим доступа: file:///D:/pz_1.html
- [4] Черемных, С. Моделирование и анализ систем. IDEF-технологии / Черемных С.В., Семенов И.О., Ручкин В.С, Москва: Россия. – 2006. – 192 с.
- [5] Батин, Н.В. Проектирование баз данных / Батин Н.В., Минск: Беларусь -2007. – 56 с.
- [6] Базы данных [Электронный ресурс]. – Режим доступа: file:///D:/EORD_bazy_dannykh.html

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг основных элементов

Файл Database.java:

```
package com.Server.dataBase;

import com.Server.constants.Constants;
import com.example.it.model.Otchet;
import com.example.it.model.Project;
import com.example.it.model.User;
import com.example.it.model.Admin;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class Database {

    private static Connection connection;
    private static Statement statement;

    public Database() {
        connectionToDB();
    }

    public void connectionToDB() {
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {

            e.printStackTrace();
        }
        try {
            connection = DriverManager.getConnection(Constants.HOST_DATABASE +
            Constants.NAME_DATABASE,
                Constants.USER_DATABASE,
                Constants.PASSWORD_DATABASE);
            statement = connection.createStatement();

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

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

    public ResultSet getUserAuthorization(User user) { //проверка на существование пользователя

        ResultSet resSet = null;
```

```

        String select = "SELECT " + Constants.LOGIN + "," + Constants.PASSWORD + " FROM " +
Constants.USERS_TABLE +
        " WHERE " + Constants.LOGIN + "=? " + "AND " + Constants.PASSWORD + "=? ";

        try {
            PreparedStatement preparedStmt = connection.prepareStatement(select);
            preparedStmt.setString(1, user.getLogin());
            preparedStmt.setString(2, user.getPassword());

            resSet = preparedStmt.executeQuery();

            System.out.println("Such user exists !");
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }

        return resSet;
    }

    public void getUserRegistration(User user) { //регистрация пользователя

        try {

            String select = "insert into " + Constants.USERS_TABLE + "(login, password, name, surname) "
                + "values (?, ?, ?, ?)";

            PreparedStatement preparedStmt = connection.prepareStatement(select);
            preparedStmt.setString(1, user.getLogin());
            preparedStmt.setString(2, user.getPassword());
            preparedStmt.setString(3, user.getName());
            preparedStmt.setString(4, user.getSurname());

            preparedStmt.execute();
            System.out.println("User was registred !");
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }

    public ResultSet getAdminAuthorization(Admin admin) { //проверка на существование админа

        ResultSet resSet = null;
        String select = "SELECT " + Constants.ADMIN_LOGIN + "," + Constants.ADMIN_PASSWORD + " FROM "
+ Constants.ADMIN_TABLE +
        " WHERE " + Constants.ADMIN_LOGIN + "=? " + "AND " + Constants.ADMIN_PASSWORD + "=? "
+ " ";
        try {
            PreparedStatement preparedStmt = connection.prepareStatement(select);
            preparedStmt.setString(1, admin.getLogin());
            preparedStmt.setString(2, admin.getPassword());

            resSet = preparedStmt.executeQuery();

        } catch (SQLException throwables) {

```

```

        throwables.printStackTrace();
    }

    return resSet;
}

public void addProject(Project project) { //добавление проекта
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(// позволяет вставлять значения
            " insert into "+Constants.PROJECTS_TABLE +" (id, name, customer, cost, deadline, level )"
            + " values (?, ?, ?, ?, ?, ?)")) {

        preparedStatement.setInt(1, project.getId());
        preparedStatement.setString(2, project.getName());
        preparedStatement.setString(3, project.getCustomer());
        preparedStatement.setString(4, project.getCost());
        preparedStatement.setString(5, project.getDeadline());
        preparedStatement.setString(6, project.getLevel());

        preparedStatement.execute(); //выполняем запрос

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

public List<Project> getAllProjects() { //просмотр проектов
    List<Project> projects = new ArrayList<>();

    try (Connection connection = DatabaseConnectionProvider.getConnection(); // если в скобках что-то
        указываете, то потом вызовется метод close()

        Statement statement = connection.createStatement()) { //выполняет запрос

        ResultSet resultSet = statement.executeQuery("SELECT " + Constants.PROJECT_ID + " , " +
            Constants.PROJECT_NAME + " , " + Constants.CUSTOMER + " , "
            + Constants.COST + " , " + Constants.DEADLINE + " , " + Constants.LEVEL + " FROM " +
            Constants.PROJECTS_TABLE);

        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String name = resultSet.getString("name");
            String customer = resultSet.getString("customer");
            String cost = resultSet.getString("cost");
            String deadline = resultSet.getString("deadline");
            String level = resultSet.getString("level");

            Project project = new Project(id, name, customer, cost, deadline, level);

            projects.add(project);
        }
    } catch (SQLException e) {

```

```

        e.printStackTrace();
    }

    return projects;
}

public List<Project> getAllCost() { //просмотр проектов
    List<Project> projects = new ArrayList<>();

    try (Connection connection = DatabaseConnectionProvider.getConnection(); // если в скобках что-то
указывается, то потом вызовется метод close()
        Statement statement = connection.createStatement()) { //выполняет запрос
        ResultSet resultSet = statement.executeQuery("SELECT " + Constants.PROJECT_NAME + " , "
            + Constants.COST + " FROM " + Constants.PROJECTS_TABLE);

        while (resultSet.next()) {

            String name = resultSet.getString("name");
            String cost = resultSet.getString("cost");

            Project project = new Project(name, cost);
            projects.add(project);}
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return projects;
}

public void updateProject(Project project) { //обновление проектов после редактирования
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(
            "UPDATE "+Constants.PROJECTS_TABLE+" SET name = ?, customer = ? , cost = ?, deadline =
?, level = ? WHERE id = ?")) {

        preparedStatement.setString(1, project.getName());
        preparedStatement.setString(2, project.getCustomer());
        preparedStatement.setString(3, project.getCost());
        preparedStatement.setString(4, project.getDeadline());
        preparedStatement.setString(5, project.getLevel());
        preparedStatement.setInt(6, project.getId());

        preparedStatement.execute();

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

public void deleteProjectByID(int id) { //удаление проекта
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(
            "delete from projects where id = ?")) {

```

```

        preparedStatement.setInt(1, id);

        preparedStatement.execute();

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

public void addUser (User user) { //добавление пользователя
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement("// позволяет вставлять значения
            "insert into "+Constants.USERS_TABLE +"( name, surname, login, password)"
            + " values ( ?, ?, ?, ?)")) {

        preparedStatement.setString(1, user.getName());
        preparedStatement.setString(2, user.getSurname());
        preparedStatement.setString(3, user.getLogin());
        preparedStatement.setString(4, user.getPassword());

        preparedStatement.execute();//выполняем запрос

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

public List<User> getAllUsers() { //просмотр пользователей
    List<User> users = new ArrayList<>();

    try (Connection connection = DatabaseConnectionProvider.getConnection();// если в скобках что-то
        указывается, то потом вызовется метод close()
        Statement statement = connection.createStatement()) { //выполняет запрос
        ResultSet resultSet = statement.executeQuery("SELECT " + Constants.ID + " , " + Constants.NAME + " , "
        + Constants.SURNAME + " , "
        + Constants.LOGIN + " , " + Constants.PASSWORD + " FROM " + Constants.USERS_TABLE);
        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String name = resultSet.getString("name");
            String surname = resultSet.getString("surname");
            String login = resultSet.getString("login");
            String password = resultSet.getString("password");

            User user = new User(id, name, surname, login, password);
            users.add(user);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return users;
}

```

```

public void updateUser(User user) { //обновление пользователей после редактирования
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(
            "update user set name = ?, surname = ?, login = ?, password = ? where id = ?")) {

        preparedStatement.setString(1, user.getName());
        preparedStatement.setString(2, user.getSurname());
        preparedStatement.setString(3, user.getLogin());
        preparedStatement.setString(4, user.getPassword());

        preparedStatement.execute();

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

public void deleteUserByID(int id) { //удаление пользователя
    try (Connection connection = DatabaseConnectionProvider.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(
            "delete from users where id = ?")) {

        preparedStatement.setInt(1, id);

        preparedStatement.execute();

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

}

public List<Otchet> getOtchet() { //просмотр пользователей
    List<Otchet> otchets = new ArrayList<>();

    try (Connection connection = DatabaseConnectionProvider.getConnection(); // если в скобках что-то
        указывает, то потом вызовется метод close()
        Statement statement = connection.createStatement()) { //выполняет запрос
        ResultSet resultSet = statement.executeQuery("SELECT " + Constants.OTHET_NAME + " , " +
            Constants.RISKS + " , "
            + Constants.PARTICIPANTS + " , " + Constants.OTCHET_DEADLINE + " FROM " +
            Constants.OTCHET);
        while (resultSet.next()) {
            String name = resultSet.getString("name");
            String risks = resultSet.getString("risks");
            String participants = resultSet.getString("participants");
            String deadline = resultSet.getString("deadline");

            Otchet user = new Otchet(name, risks, participants, deadline);
            otchets.add(user);
        }
    } catch (SQLException e) {

```



```

        e.printStackTrace();
    }

    return otchets;
}

public static void filewriter() throws ClassNotFoundException {
    Class.forName("org.postgresql.Driver");
    ResultSet resultSet = null;
    List<String> data = new ArrayList<String>();
    List<String> head = new ArrayList<String>();
    try (Connection connection = DatabaseConnectionProvider.getConnection();// если в скобках что-то
указывается, то потом вызовется метод close()
        Statement statement = connection.createStatement()
    ) {
        String sqlQuery = "select * from " + Constants.PROJECTS_TABLE;
        try {
            resultSet = statement.executeQuery(sqlQuery);
            while (resultSet.next()) {
                String id = String.valueOf(resultSet.getInt(1));
                String name = resultSet.getString(2);
                String customer = resultSet.getString(3);
                String cost = resultSet.getString(4);
                String deadline = resultSet.getString(5);
                String level = resultSet.getString(6);
                data.add(id + "\t" + name + "\t\t" + customer + "\t" + cost + "\t\t" + deadline + "\t\t" + level);
            }
            head.add("\n");
            head.add("ID\tName\t\tCost\t\tCustomer\t\tDeadline\t\tReadiness level\n-----");
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            System.out.println("Запрос " + sqlQuery + " был обработан!");
        }
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

public static void filewriter1() throws ClassNotFoundException {
    Class.forName("org.postgresql.Driver");
    ResultSet resultSet = null;
    List<String> data = new ArrayList<String>();
    List<String> head = new ArrayList<String>();
    try (Connection connection = DatabaseConnectionProvider.getConnection();// если в скобках что-то
указывается, то потом вызовется метод close()
        Statement statement = connection.createStatement()
    ) {
        String sqlQuery = "select * from " + Constants.OTCHET;

```

```

try {
    resultSet = statement.executeQuery(sqlQuery);
    while (resultSet.next()) {

        String name = resultSet.getString(1);
        String risks =resultSet.getString(2);
        String participants = resultSet.getString(4);
        String deadline = resultSet.getString(4);

        data.add(name + "\t\t" +risks + "\t\t" + participants + "\t\t\t" + deadline);
    }
    head.add("\n");
    head.add("Name\t\t\tRisks\t\t\tParticipants\t\t\tDeadline\n-----");
-----");
    writeToFile(head, "UserOtchet.txt");
    writeToFile(data, "UserOtchet.txt");
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    System.out.println("Запрос " + sqlQuery + " был обработан!");
}
} catch (SQLException throwables) {
    throwables.printStackTrace();
}
}

private static void writeToFile(java.util.List<String> list, String path) {
    BufferedWriter out = null;
    try {
        File file = new File(path);
        out = new BufferedWriter(new FileWriter(file, true));
        for (String s : list) {
            out.write(s);
            out.newLine();
        }
        out.close();
    } catch (IOException e) {
    }
}

}

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг скрипта генерации базы данных

Файл IT.sql:

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 13.5
-- Dumped by pg_dump version 13.5

-- Started on 2021-12-09 16:44:55

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

--
-- TOC entry 2 (class 3079 OID 16384)
-- Name: adminpack; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS adminpack WITH SCHEMA pg_catalog;

--
-- TOC entry 3012 (class 0 OID 0)
-- Dependencies: 2
-- Name: EXTENSION adminpack; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION adminpack IS 'administrative functions for PostgreSQL';

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- TOC entry 204 (class 1259 OID 32825)
-- Name: admin; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.admin (
    id integer NOT NULL,
    login character varying NOT NULL,
    password character varying NOT NULL
);

ALTER TABLE public.admin OWNER TO postgres;

--
-- TOC entry 203 (class 1259 OID 32823)
-- Name: admin_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.admin ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
```

```

SEQUENCE NAME public.admin_id_seq
START WITH 1
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
CACHE 1
);

--
-- TOC entry 207 (class 1259 OID 32859)
-- Name: otchet; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.otchet (
    name character varying(50) NOT NULL,
    risks character varying(50),
    participants character varying(100),
    deadline character varying(50)
);

ALTER TABLE public.otchet OWNER TO postgres;

--
-- TOC entry 206 (class 1259 OID 32853)
-- Name: projects; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.projects (
    id integer NOT NULL,
    name character varying(50) NOT NULL,
    customer character varying(50),
    cost character varying(50),
    deadline character varying(50),
    level character varying(50)
);

ALTER TABLE public.projects OWNER TO postgres;

--
-- TOC entry 205 (class 1259 OID 32851)
-- Name: projects_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

CREATE SEQUENCE public.projects_id_seq
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;

ALTER TABLE public.projects_id_seq OWNER TO postgres;

--
-- TOC entry 3013 (class 0 OID 0)
-- Dependencies: 205
-- Name: projects_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: postgres
--

ALTER SEQUENCE public.projects_id_seq OWNED BY public.projects.id;

--
-- TOC entry 202 (class 1259 OID 24626)
-- Name: users; Type: TABLE; Schema: public; Owner: postgres

```

```

--

CREATE TABLE public.users (
    id integer NOT NULL,
    name character varying(50) NOT NULL,
    surname character varying(50) NOT NULL,
    login character varying(50) NOT NULL,
    password character varying(50) NOT NULL
);

ALTER TABLE public.users OWNER TO postgres;

--
-- TOC entry 201 (class 1259 OID 24624)
-- Name: users_id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public.users ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
    SEQUENCE NAME public.users_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);

--
-- TOC entry 2871 (class 2606 OID 32832)
-- Name: admin admin_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.admin
    ADD CONSTRAINT admin_pkey PRIMARY KEY (id);

--
-- TOC entry 2875 (class 2606 OID 32863)
-- Name: otchet otchet_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.otchet
    ADD CONSTRAINT otchet_pkey PRIMARY KEY (name);

--
-- TOC entry 2873 (class 2606 OID 32858)
-- Name: projects projects_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.projects
    ADD CONSTRAINT projects_pkey PRIMARY KEY (name);

--
-- TOC entry 2869 (class 2606 OID 24630)
-- Name: users users_pkey; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.users
    ADD CONSTRAINT users_pkey PRIMARY KEY (id);

--
-- TOC entry 2876 (class 2606 OID 32864)
-- Name: otchet otchet_name_fkey; Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.otchet
    ADD CONSTRAINT otchet_name_fkey FOREIGN KEY (name) REFERENCES public.projects(name);

```

Графический материал

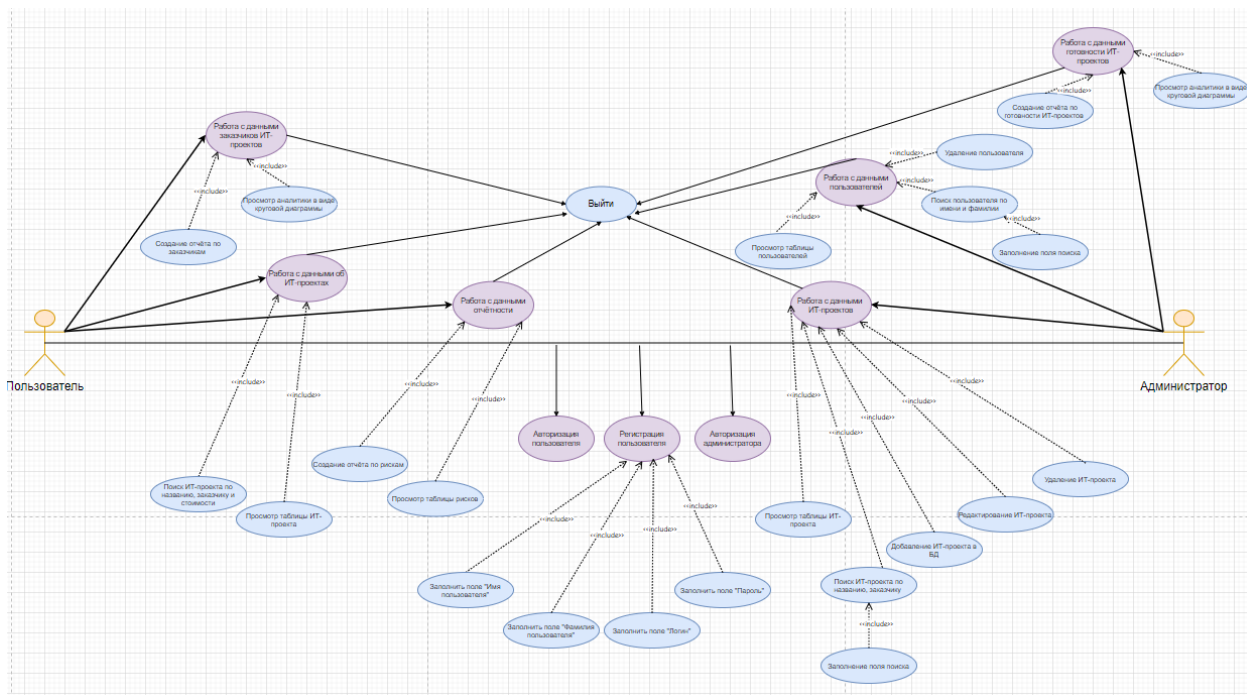


Рисунок В.1 – Диаграмма вариантов использований

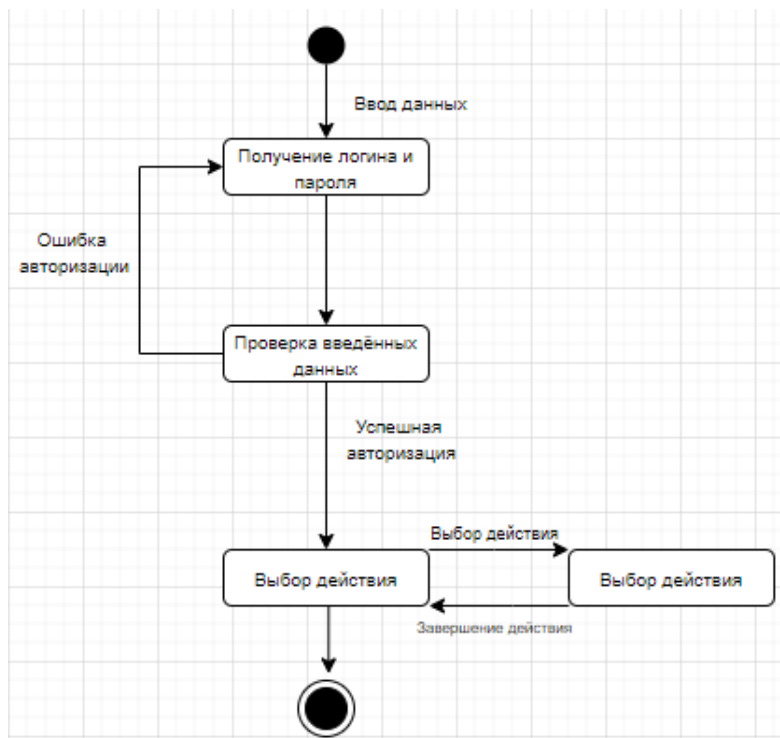


Рисунок В.2 – Диаграмма состояний

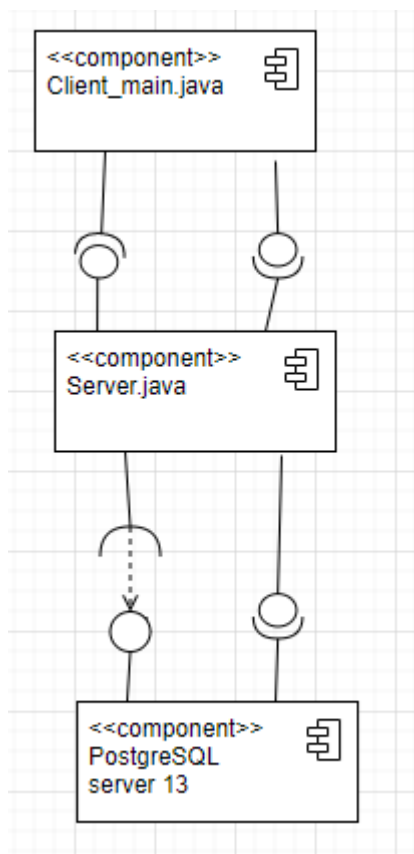


Рисунок В.3 – Диаграмма компонентов

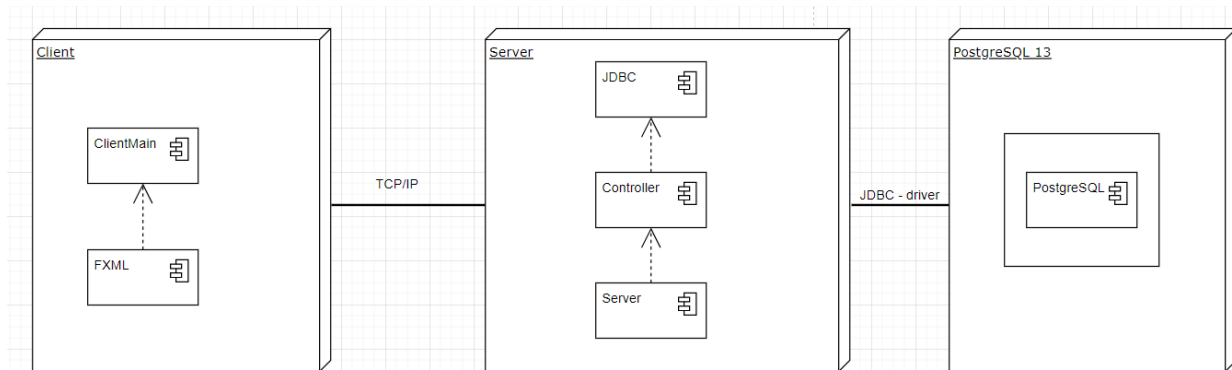


Рисунок В.4 – Диаграмма развертывания

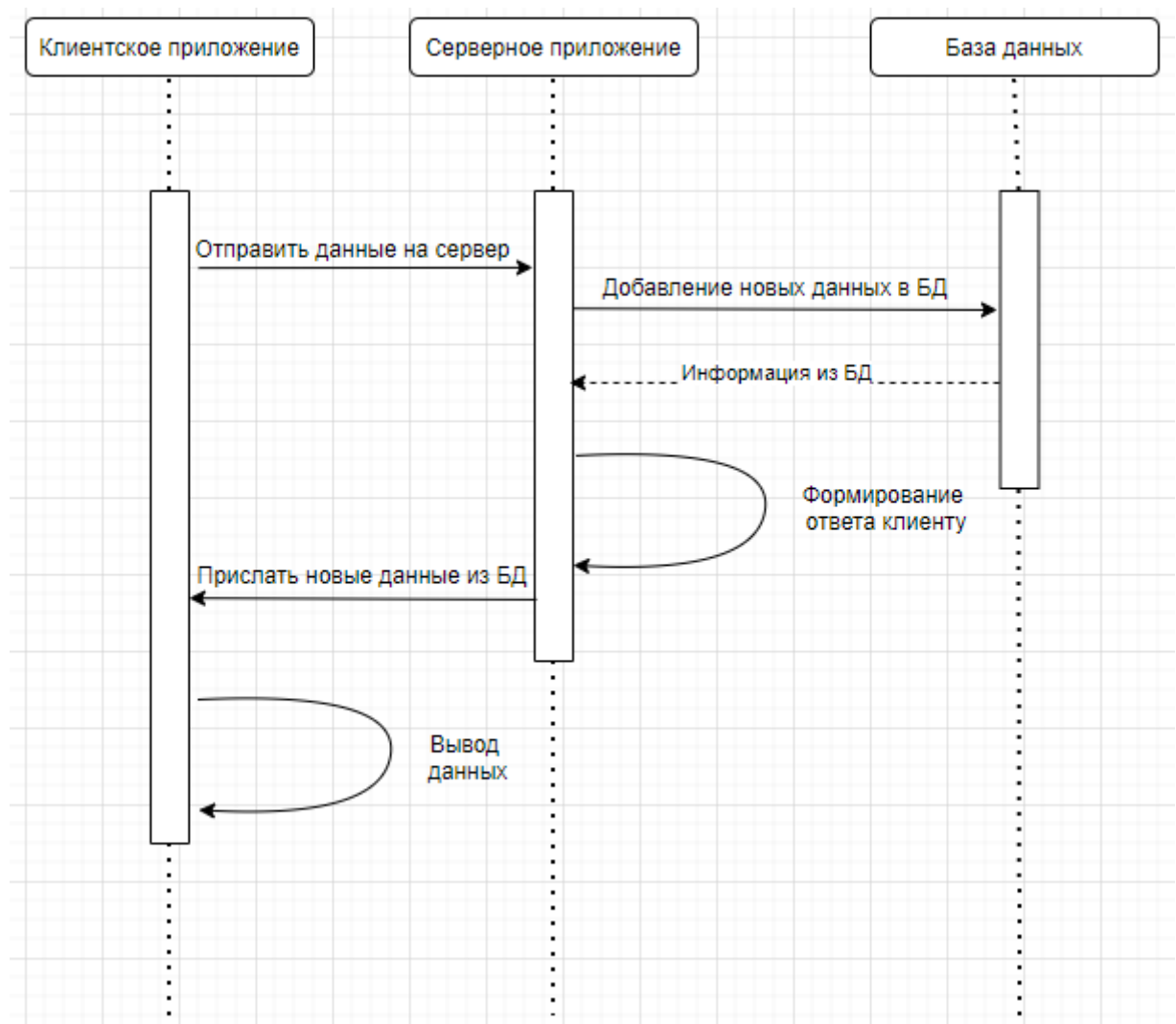


Рисунок В.5 – Диаграмма последовательностей

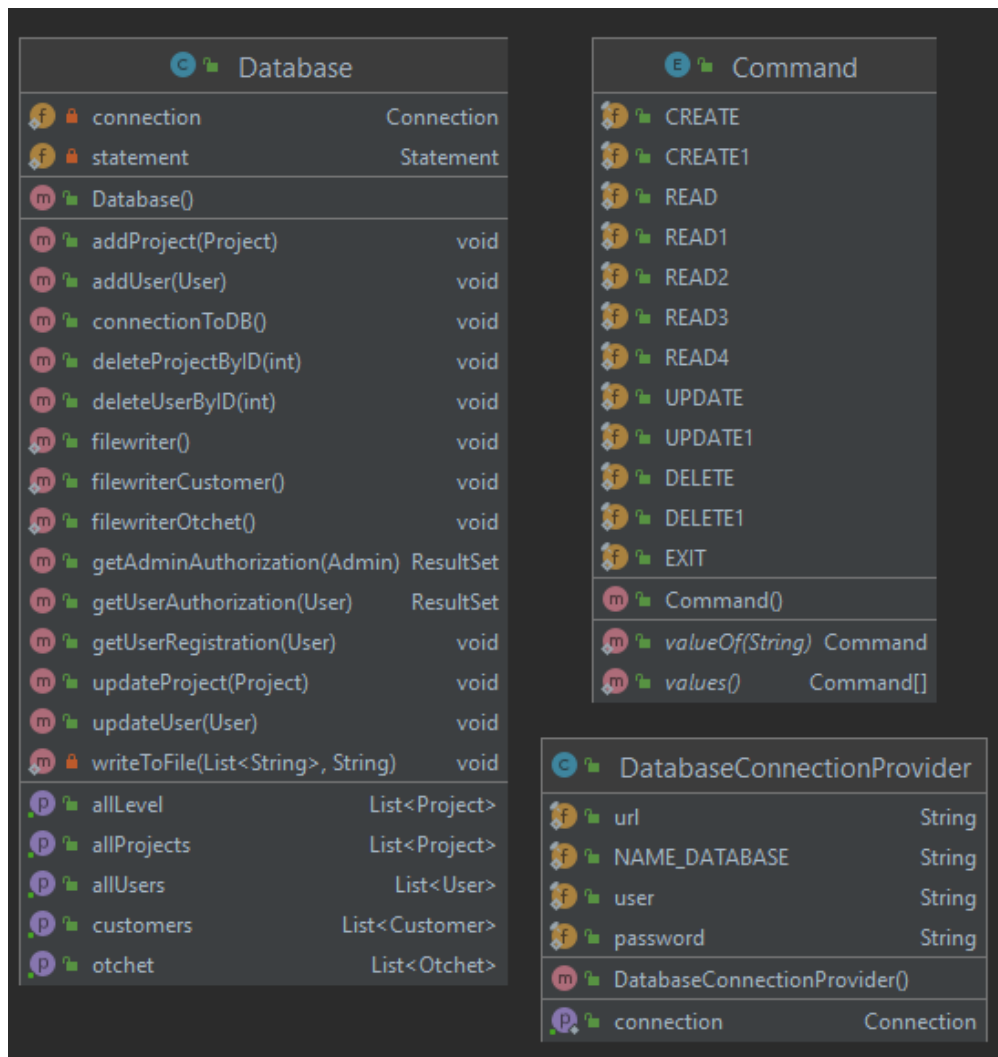


Рисунок В.6 – Диаграмма классов серверной части

ПРИЛОЖЕНИЕ Б

Перв. примен. ГУИР. Д	Зона	Обозначение	Наименование	Дополнительные сведения
Справочный №			<u>Текстовые документы</u>	
		БГУИР КР 1–40 05 01-10 010 ПЗ	Пояснительная записка	60 с.
			<u>Графические документы</u>	
		ГУИР.506196.010Д1	Диаграмма прецедентов	Формат А4
		ГУИР.506196.010Д2	Диаграмма классов	Формат А4
		ГУИР.506196.010Д3	Структура графического пользовательского интерфейса	Формат А3
		ГУИР.506196.010Д4	Диаграмма компонентов	Формат А4
ПоКПись и				
		ГУИР.506196.010Д6	Диаграмма развертывания	Формат А4
		ГУИР.506196.010Д7	Диаграмма состояний	Формат А4
Инв. №				
		ГУИР.506196.010Д8	Информационная модель	Формат А4
Взам. Инв.				
		ГУИР.506196.010Д9	Схема алгоритма	Формат А4
ПоКПись и				
		ГУИР.506196.010Д10	Функциональная модель	Формат А4
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				
Инв. №				

