Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Дисциплина: Объектно-ориентированно программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

СИСТЕМА УПРАВЛЕНИЯ СПА-КОМПЛЕКСОМ

БГУИР КР 6-05-0612-03 158 ПЗ

Студент К.А. Хаджинова
Руководитель А.К. Ючков

Минск 2024

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись)

2024 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Хаджиновой Ксении Александровне*

1. Тема проекта Система управления СПА-комплексом
2. Срок сдачи студентом законченного проекта *18 декабря 2024 г.*
3. Исходные данные к проекту: *В базе данных системы управления СПА-комплексом содержится информация об услугах, сотрудниках, клиентах, резервациях процедур, оплате услуг.*
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.
1. Анализ предметной области.
2. Проектирование информационной ситемы кинотеатра.
3. Разработка информационной ситемы кинотеатра.
Заключение
Список использованных источников
Приложения
Ведомость курсового проекта

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

Диаграмма активностей системы управления СПА-комплексом (формат А4)

6. Консультант по проекту (с обозначением разделов проекта) _____
А. К. Ючков

7. Дата выдачи задания *12 сентября 2024 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 14.10 – 30%

раздел 2 к 11.11 – 60%

раздел 3 к 09. 12 – 90%

оформление пояснительной записки и графического материала к 16.12 – 100 %

Защита курсового проекта с 18.12

Руководитель курсового проекта _____ *А. К. Ючков*
(подпись)

Задание принял к исполнению _____ *К. А. Хаджинова*
(подпись)

СОДЕРЖАНИЕ

Введение 7
1 Анализ предметной области 9
1.1 Описание объекта автоматизации 9
1.2 Постановка задачи 10
2 Проектирование системы управления СПА-комплексом 12
2.1 Разработка функционала системы 12
2.2 Разработка иерархии классов 17
3 Разработка системы управления СПА-комплексом 20
4 Руководство пользователя 26
Заключение 30
Список использованных источников 32
Приложение А (обязательное) Программный код 33
Ведомость курсовой работы 59

ВВЕДЕНИЕ

В наше время СПА-комплексы становятся все более популярными местами для отдыха и оздоровления. Эти учреждения предлагают уникальную атмосферу, в которой клиенты могут не только расслабиться, но и восстановить физическое и эмоциональное состояние. СПА-комплекс предлагает разнообразные процедуры, такие как классические массажи, уходы за кожей, ароматерапия и грязелечение, которые помогают людям справляться со стрессом, улучшают общее самочувствие и способствуют укреплению здоровья.

Современный СПА-комплекс невозможно представить без информационной системы, автоматизирующей множество процессов и облегчающей жизнь как работникам, так и посетителям. Информационная система управления СПА-комплексом играет ключевую роль в организации всех операций, связанных с предоставлением услуг. Это программное обеспечение включает функционал для записи клиентов на процедуры, управления ассортиментом услуг, расчетов и обработки платежей. Благодаря этой системе работники могут легко и быстро получать информацию о доступных услугах, занятости сотрудников и свободных временных интервалах. Это позволяет администратору оперативно реагировать на запросы клиентов, увеличивая скорость обслуживания и повышая качество клиентского сервиса.

Посетители могут самостоятельно записываться на процедуры через онлайн-платформу, что снижает нагрузку на администраторов и улучшает клиентский опыт. Возможность онлайн-записи позволяет клиентам выбирать удобное время и получать актуальную информацию о доступных процедурах и ценах, что уменьшает вероятность ошибок при записи и делает процесс более надежным.

Разработка данной информационной системы актуальна по нескольким причинам. Во-первых, она способствует ускорению и упрощению процесса записи на процедуры, что особенно важно в условиях высокого потока клиентов. Во-вторых, система делает услуги более доступными, позволяя клиентам получать информацию и записываться на процедуры в любое время и из любого места. Широкий функционал таких систем включает возможность просмотра истории посещений, управления личными данными и получения уведомлений о предстоящих процедурах. Это делает использование системы более удобным и информативным, повышая уровень удовлетворенности клиентов.

Использование объектно-ориентированного программирования (ООП) при создании информационных систем на основе классовой иерархии имеет множество преимуществ. ООП позволяет уменьшить сложность системы,

структурировать информацию и сделать её более управляемой. Программа представляется как совокупность объектов, каждый из которых является экземпляром определённого класса, что упрощает поддержку и развитие системы. Гибкость и адаптируемость системы к изменениям особенно важны в динамичной сфере услуг, где требования клиентов могут меняться.

Система управления СПА-комплексом включает несколько ключевых компонентов. Управление услугами позволяет администратору добавлять и редактировать предложения, их описания, цены и длительность. Учет сотрудников обеспечивает возможность отслеживания графиков работы, специализаций и занятости каждого работника, что позволяет эффективно распределять рабочие часы. Регистрация и управление данными клиентов включает создание профилей, содержащих информацию о предыдущих посещениях и предпочтениях. Управление записями клиентов на услуги позволяет удобно отменять или изменять записи, а система уведомляет клиентов о предстоящих процедурах. Обработка платежей включает учёт различных способов оплаты, что делает процесс удобным для клиентов и уменьшает время обслуживания.

Целью курсового проекта является разработка системы управления СПА-комплексом, которая предоставит пользователям все вышеперечисленные возможности. Основными задачами проекта являются разработка информационной системы управления и её реализация в виде программы на языке *C++*. Также важной частью проекта является оформление текстовой и графической документации в соответствии с требованиями государственных стандартов и стандартов предприятия [1-2]. В рамках проекта планируется создание диаграмм для визуализации структуры системы, включая диаграмму случаев использования, которая описывает взаимодействие пользователей с системой, классовую диаграмму, демонстрирующую основные классы и их взаимосвязи, и диаграмму активности, иллюстрирующую последовательность действий пользователей при работе с системой.

Создание продуманной иерархии классов и применение принципов объектно-ориентированного программирования являются основой для разработки качественной информационной системы управления СПА-комплексом. Это позволит удовлетворить нужды пользователей и способствовать оптимизации процессов, улучшая качество предоставляемых услуг. В конечном итоге, разработанная система станет важным инструментом для успешной работы СПА-комплекса, способствуя его развитию и повышению конкурентоспособности на рынке. Инвестиции в такую информационную систему обеспечат долгосрочные преимущества как для бизнеса, так и для клиентов, создавая положительный имидж и увеличивая уровень доверия к услугам СПА-комплекса.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание объекта автоматизации

Объектом автоматизации в данной курсовой работе является СПА-комплекс, для которого необходимо разработать информационную систему. Эта система представляет собой важный инструмент, обеспечивающий эффективное функционирование СПА-комплекса. В современных условиях наличие информационной системы становится обязательным атрибутом любого СПА-учреждения. Без такой системы клиентам сложно получить актуальную информацию о доступных услугах, ценах и расписании процедур.

В условиях быстрого темпа жизни и высоких ожиданий клиентов, необходимость в автоматизации процессов становится особенно актуальной. Информационная система позволяет клиентам записываться на процедуры из дома, что значительно упрощает процесс выбора и бронирования услуг. Это избавляет их от необходимости звонить в СПА или приходить на место для получения информации о доступных процедурах. Система предоставляет возможность выбрать удобную дату и время, что делает взаимодействие с СПА-комплексом более комфортным и доступным.

Кроме того, информационная система облегчает работу сотрудников СПА-комплекса, позволяя им эффективно управлять резервациями и контролировать занятость персонала. Система обеспечивает наличие свободных временных интервалов для записи клиентов, что помогает избежать ошибок, таких как двойная запись на одну и ту же процедуру. Это особенно важно для поддержания высокого уровня обслуживания и удовлетворенности клиентов.

Информационная система также включает в себя базу данных с информацией обо всех услугах и прейскурантах. Администраторы СПА могут редактировать эту базу данных, добавляя новые услуги или изменяя существующие. Это позволяет быстро реагировать на изменения в спросе и адаптироваться к потребностям клиентов. База данных также может содержать информацию о клиентах, их предпочтениях и истории посещений, что помогает персонализировать услуги и улучшить клиентский опыт.

Разработка информационной системы для управления СПА-комплексом направлена на создание удобного инструмента как для клиентов, так и для сотрудников. Она обеспечивает эффективное взаимодействие между всеми участниками процесса, минимизируя вероятность ошибок и повышая общую эффективность работы комплекса.

1.2 Постановка задачи

В результате выполнения курсового проекта был проведён детальный анализ выбранной предметной области, на основе которого были сформулированы требования к информационной системе и поставлена задача. В соответствии с техническим заданием была разработана иерархия классов для информационной системы СПА-комплекса.

В процессе изучения предметной области сделаны выводы о значимости СПА-комплексов в жизни человека. В современном мире, когда существуют передовые технологии и программное обеспечение, сложно представить современный СПА без информационной системы, что подчеркивает необходимость такой системы.

В ходе исследования и сравнения систем-аналогов были рассмотрены две информационные системы, предоставляющие услуги в области СПА. Изучение функционала этих систем и их основных возможностей позволило дополнить и обновить сформулированные требования к разрабатываемой системе.

Для разработки информационной системы использовался язык программирования *C++*, который поддерживает объектно-ориентированный подход. Была создана иерархия классов, в которой был разработан базовый абстрактный класс *User*, от которого наследуются классы *Guest* и *Staff*. Каждый из пользователей информационной системы обладает своим функционалом, позволяющим взаимодействовать с системой, а работники СПА, особенно администратор, способны управлять ею [3].

В процессе реализации программы были использованы все четыре основных принципа объектно-ориентированного программирования (ООП), среди которых важное место занимали наследование и полиморфизм, реализованный с использованием виртуальных функций.

Проектирование системы основывалось на диаграммах: диаграмме вариантов использования (*use case diagram*), диаграмме активности (*activity diagram*) и диаграмме классов (*classes diagram*). Диаграмма вариантов использования иллюстрирует основные функции системы, такие как просмотр доступных процедур, бронирование и управление пользователями. Диаграмма активности показывает последовательность действий пользователей при взаимодействии с системой, что помогает понять логику работы приложения. Диаграмма классов демонстрирует структуру классов и их взаимосвязи, что обеспечивает четкое понимание архитектуры системы.

Для работы с базой данных была подключена MySQL. Это позволило эффективно управлять данными о пользователях, процедурах и записях на

процедуры. Программа взаимодействует с базой данных через SQL-запросы с использованием подготовленных выражений (prepared statements), что обеспечивает безопасность операций с данными и защиту от SQL-инъекций. Система включает функции для выполнения основных операций: добавления новых пользователей, управления процедурами и создания бронирований.

Данный проект информационной системы для СПА-комплекса направлен на создание удобного инструмента как для клиентов, так и для сотрудников. Это повысит общую эффективность работы комплекса и улучшит качество обслуживания. Разработанная система имеет потенциал для дальнейшего развития и адаптации к меняющимся требованиям рынка услуг СПА.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ СПА-КОМПЛЕКСОМ

2.1 Разработка функционала системы

Разработка функционала информационной системы для СПА-комплекса включает создание четырех типов пользователей: клиента, администратора, кассира и специалиста (например, массажиста или косметолога). Каждый тип пользователя имеет свои уникальные функции и возможности, соответствующие их ролям в системе.

Для реализации данной системы был выбран язык программирования *C++*, который поддерживает объектно-ориентированный подход. Это позволяет создавать гибкие и расширяемые структуры классов. В коде использовались следующие библиотеки:

– *iostream*: для ввода и вывода данных;
– *iomanip*: для форматирования вывода;
– *string*: для работы со строками;
– *vector*: для использования динамических массивов;
– *memory*: для управления памятью с помощью умных указателей;
– *regex*: для валидации *email* адресов;
– *mysql_driver.h, mysql_connection.h, cppconn/prepared_statement.h, cppconn/resultset.h*: для работы с базой данных *MySQL*.

Использование *MySQL* в качестве системы управления базами данных (СУБД) позволяет эффективно хранить и обрабатывать информацию о пользователях, процедурах и записях на процедуры [4]. *MySQL* была выбрана из-за своей производительности, надежности и поддержки большого объема данных.

В системе используется *MySQL* для хранения информации о пользователях, процедурах и записях на процедуры. Это позволяет обеспечить надежное управление данными и их структурирование. При этом работа с базой данных осуществляется через *SQL*-запросы с использованием подготовленных выражений (*prepared statements*), что обеспечивает безопасность операций с данными и защиту от *SQL*-инъекций.

В системе выделяются несколько групп методов, которые обеспечивают выполнение ключевых операций:

– методы для работы с процедурами: включают добавление, редактирование и удаление процедур;
– методы для работы с расписанием: позволяют управлять доступными временными слотами и записываться на процедуры;

– методы для работы с клиентами: обеспечивают регистрацию новых клиентов, редактирование их данных и просмотр истории посещений;

– методы для работы с отчетами: позволяют создавать отчеты о проведенных процедурах и финансовых поступлениях;

– методы для работы с пользователями: управляют правами доступа и данными сотрудников СПА.

Алгоритм работы программы можно описать следующими шагами:

1. При запуске программы открывается страница авторизации, где пользователю предлагаются варианты: вход в систему (авторизация), регистрация или выход из системы.

2. При выборе варианта регистрации пользователь может зарегистрироваться как клиент или специалист. Если выбран вход, пользователю предоставляется возможность ввести логин и пароль. При корректном вводе происходит вход в личный кабинет, где открывается соответствующее меню для каждого типа пользователя. В случае неправильного ввода система сообщает об ошибке авторизации и возвращает пользователя на страницу авторизации.

3. В пользовательском меню текущему пользователю предоставляется функционал в зависимости от его роли. Например, администратор может просматривать информацию о других пользователях системы (клиентах и специалистах) или объектах базы данных СПА (например, процедурах или резервациях). Для этого администратор получает доступ к соответствующему файлу с данными.

4. При выборе варианта добавления нового объекта базы данных (например, новой процедуры или резервации) открывается соответствующий файл для записи данных.

Код программы реализует различные функции для управления процедурами и бронированиями. Например, функция *printTable* отвечает за форматирование вывода данных в виде таблицы, что делает интерфейс более удобным для пользователя. Вот классы пользователей в информационной системе реализуют методы, которые позволяют выполнять операции в зависимости от их ролей:

– класс *Guest* предоставляет возможность просматривать забронированные процедуры через метод *viewProcedures*, записываться на процедуры с помощью метода *bookProcedure* и управлять своими бронированиями с помощью метода *manageBookings*;

– класс *Manager* включает методы, которые позволяют управлять пользователями и процедурами. Метод *manageUsers* дает возможность добавлять, просматривать и удалять пользователей, тогда как метод *manageProcedures* позволяет менеджерам управлять процедурами, включая их просмотр, добавление и удаление;

– класс *Staff* предоставляет функционал для управления бронированиями. Метод *manageBookings* позволяет сотрудникам просматривать все бронирования и изменять их статус.

Рассмотрим пример реализации функций. Функция *bookProcedure* в классе *Guest* отвечает за процесс бронирования. Сначала она получает доступные процедуры из базы данных, а затем выводит список этих процедур пользователю. После этого запрашивается *ID* процедуры для бронирования; система проверяет корректность введенного *ID*. Если *ID* действителен, происходит вставка записи о бронировании в базу данных. Функция *manageBookings* в классе *Staff* позволяет просматривать все бронирования и изменять их статус. Она извлекает данные о всех бронированиях из базы данных и выводит список текущих бронирований. Сотрудник может выбрать конкретное бронирование для изменения статуса, например, подтвердить или отменить его. Каждый метод включает обработку исключений для обработки ошибок *SQL*-запросов, что обеспечивает надежность работы системы.

Проектирование системы для СПА-комплекса основывалось на диаграммах, включая диаграмму вариантов использования (*use case*), диаграмму активности и диаграмму классов [6]. Диаграмма вариантов использования иллюстрирует основные функции системы, такие как просмотр доступных процедур, бронирование процедур, управление пользователями и отчетами. Она помогает определить взаимодействие между различными типами пользователей, такими как гости, администраторы и персонал, и системой. Диаграмма активности показывает последовательность действий пользователей при взаимодействии с системой, описывая процесс авторизации пользователя, выбор процедур, управление записями и выход из системы. Диаграмма классов демонстрирует структуру классов системы и их взаимосвязи, показывая наследование между классами пользователей (*User, Guest, Manager, Staff*).

Разрабатываемая информационная система обеспечит удобное взаимодействие пользователей с комплексом, автоматизируя ключевые процессы и повышая эффективность работы. Каждый пользователь имеет свой вариант использования системы, обусловленный его функционалом. Для наглядности была разработана диаграмма вариантов использования пользователей, представленная на рисунке 2.1.

Диаграмма разработана с помощью *PlantUML* [7], вот код данной диаграммы:

```
@startuml
!theme plain
skinparam linetype ortho
```

```
skinparam ArrowThickness 1
skinparam componentStyle rectangle
skinparam usecase {
   BackgroundColor #E3F2FD
   BorderColor #90CAF9
}

actor "Гость" as guest
actor "Персонал" as staff
actor "Управляющий" as manager

rectangle "Система управления СПА комплексом" {
   usecase "Просмотреть\nдоступные процедуры" as UC3


   usecase "Забронировать\nпроцедуру" as UC1


   usecase "Отменить\nбронь" as UC2


   usecase "Управление\nпроцедурами" as UC4


   usecase "Управление\nбронированиями" as UC5


   usecase "Управление\nпользователями" as UC6
}

'Расположение акторов
guest --> UC1
guest --> UC2
guest --> UC3

staff --> UC1
staff --> UC2
staff --> UC4
```

manager --> UC4
manager --> UC5
manager --> UC6

' Наследование
staff --|> guest : <<inherits>>
manager --|> staff : <<inherits>>
manager --|> guest : <<inherits>>

UC1 ..> UC3 : <<include>>
UC4 ..> UC3 : <<include>>
UC2 ..> UC1 : <<extend>>
UC5 ..> UC1 : <<extend>>

@enduml

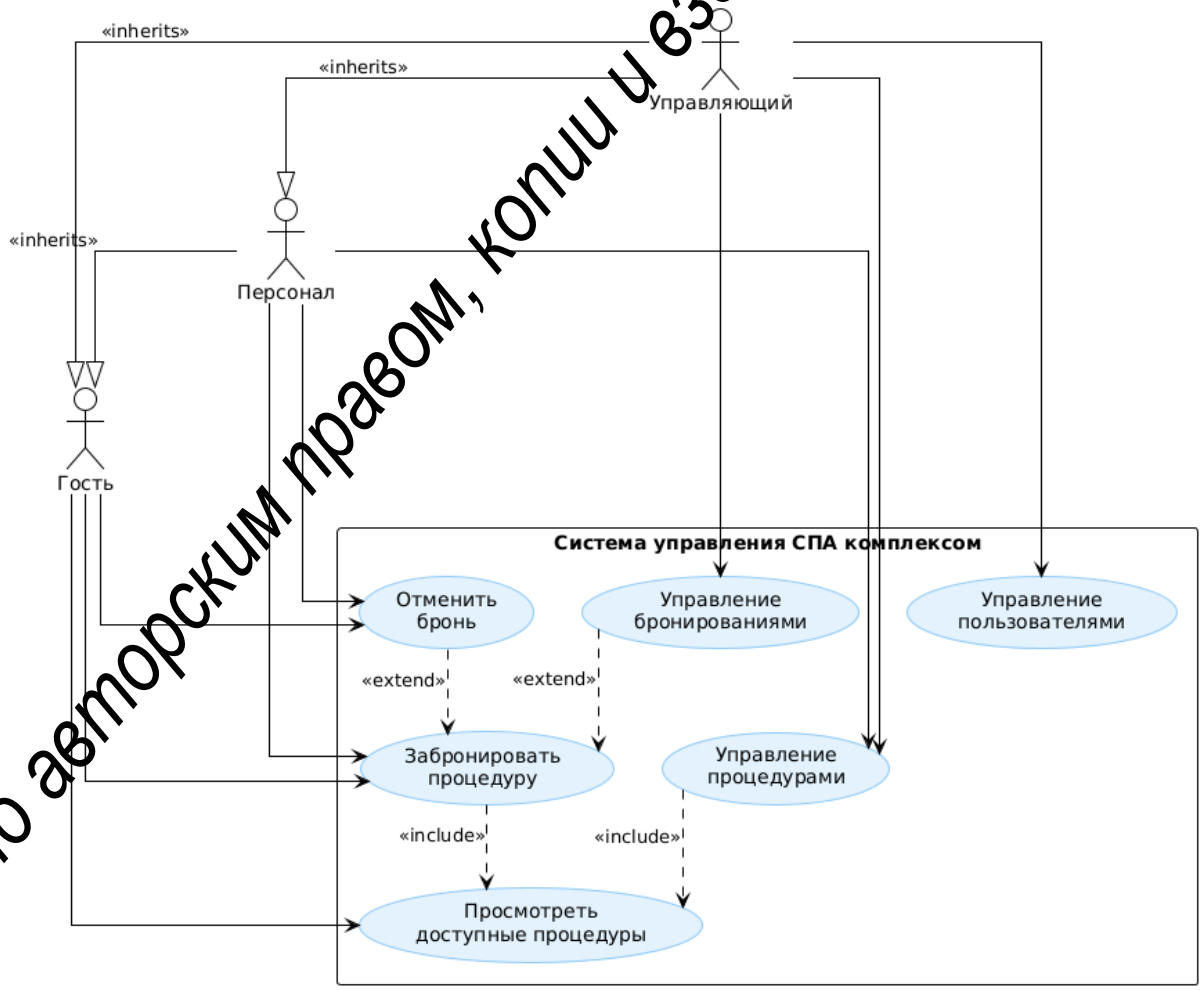

Рисунок 2.1 – Диаграмма вариантов использования системы

2.2 Разработка иерархии классов

При разработке информационной системы было создано тринадцать классов и три типа интерфейса, предназначенные для различных пользователей, реализованные через функции меню [8-9].

Абстрактный класс *User* служит основой для всех пользователей системы. Класс *Guest*, производный от *User*, представляет клиента и обладает базовым функционалом. Класс *Manager* предоставляет администратору возможности управления системой, включая добавление и редактирование процедур.

Все пользователи системы имеют общие поля: логин, пароль, имя и контактная информация, которые вынесены в абстрактный класс *User*. Методы класса *User*, такие как *viewProcedures()* и *viewBookings()* обеспечивают доступ к основным функциям системы.

Более подробное описание классов с их полями и методами представлено в диаграмме иерархии классов на рисунке 2.1.

Данная диаграмма также была спроектирована в *PlantUML*. Код диаграммы:

```
@startuml
abstract class "Пользователь" {
   +имя: String
   +email: String
   +пароль: String
   +войти(): void
   +выйти(): void
}

class "Гость" {
   +забронироватьПроцедуру(): void
   +отменитьБронь(): void
   +просмотретьПроцедуры(): void


class "Управляющий" {
   +управлятьПроцедурами(): void
   +управлятьПользователями(): void
   +настроитьРасписание(): void
}
```

```
class "Персонал" {
   +предоставитьУслугу(): void
   +просмотретьБронирования(): void
}

class "Процедура" {
   +название: String
   +цена: float
   +описание: String
}

class "Бронирование" {
   +дата: Date
   +время: Time
   +статус: String
   +изменитьСтатус(): void
}

class "Расписание" {
   +дата: Date
   +время: Time
}

class "КонтактнаяИнформация" {
   +телефон: String
   +адрес: String
}

' Наследование
Пользователь <|-- Гость
Пользователь <|-- Управляющий
Пользователь <|-- Персонал

' Ассоциации
Управляющий --> Процедура : "управляет"
Управляющий --> Пользователь : "управляет"
Гость --> Бронирование : "создает"
```

' Композиция и агрегация
Процедура "1" -- "0..*" Бронирование : "имеет"*
Расписание "1" -- "1" Процедура : "назначает"
КонтактнаяИнформация "1" o-- "1" Пользователь : "имеет"
@enduml

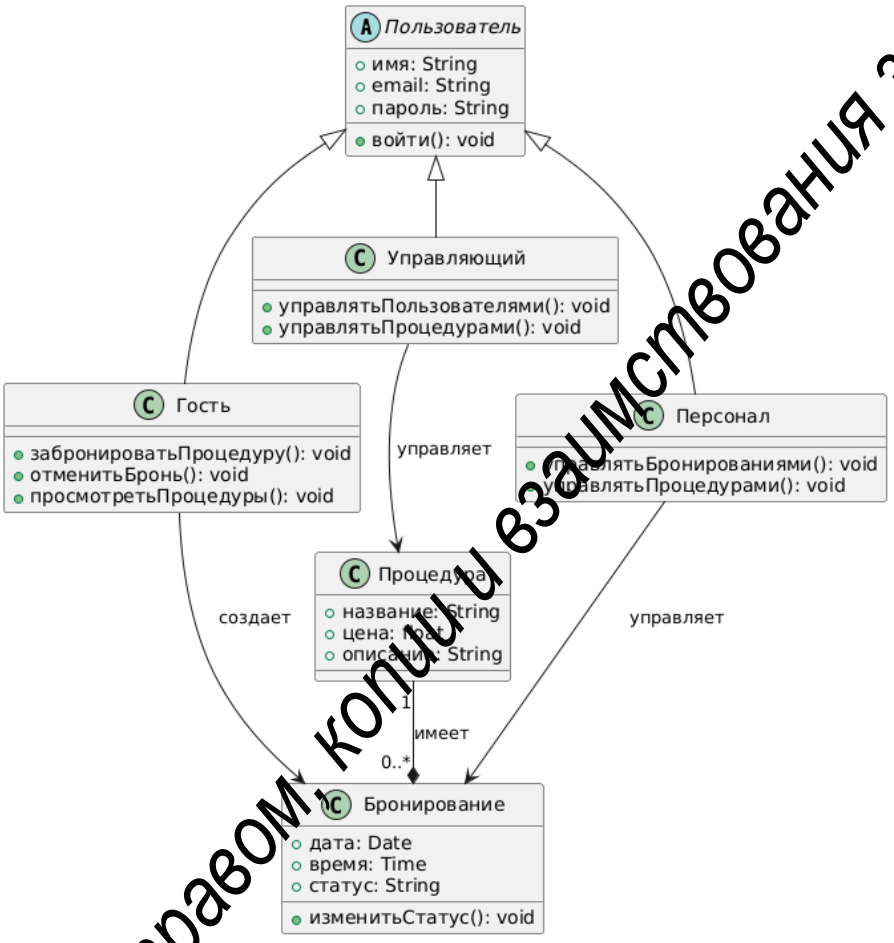

Рисунок 2.2 – Иерархия классов

На диаграмме показаны основные классы, их атрибуты и методы, а также связи между классами, включая наследование и ассоциации.

3 РАЗРАБОТКА СИСТЕМЫ УПРАВЛЕНИЯ СПА-КОМПЛЕКСОМ

Процесс реализации программы начался с создания классов, необходимых для базовой работы системы. В первую очередь были разработаны классы пользователей, которые обеспечивают минимальный функционал системы. Важным этапом стало создание абстрактного класса *User*, который стал основой для всех пользователей. Этот класс включает в себя общие атрибуты, такие как имя, *email* и пароль, а также методы для аутентификации, что обеспечивает единообразие в управлении пользователями.

Особое внимание было уделено разработке класса *Manager* и его методов, так как их некорректная работа могла привести к сбоям в функционировании всей системы. Класс *Manager* отвечает за управление процедурами и пользователями, что критически важно для успешного функционирования системы. Реализация методов, позволяющих добавлять, удалять и модифицировать записи о процедурах и пользователях, потребовала тщательной проработки логики и взаимодействия с базой данных.

После успешного завершения этапа разработки класса *Manager* были реализованы другие классы, такие как *Guest* и *Staff*. Класс *Guest* предоставляет функционал для взаимодействия с системой, включая методы для бронирования процедур, отмены броней и просмотра доступных процедур. Это важно для создания удобного интерфейса для конечного пользователя. Класс *Staff*, в свою очередь, отвечает за выполнение услуг и управление записями, что позволяет эффективно организовать рабочий процесс.

На следующем этапе были добавлены методы для работы с записями на процедуры. Ключевыми методами стали *bookProcedure* в классе *Guest*, который позволяет клиентам записываться на процедуры, а также методы *viewBookings* в классе *Manager* и *viewProcedures* в классе *Guest*, которые дают возможность просматривать записи. Наиболее трудоемкой задачей оказалась реализация метода *bookProcedure*, так как он требует взаимодействия с несколькими файлами и включает в себя сложную логику проверки доступности процедур, обработки дат и времени, а также обновления записей в базе данных.

На заключительном этапе реализации были добавлены дополнительные методы для взаимодействия пользователей с информационной системой. Для хранения данных использовался *MySQL*.

Скрипт по созданию таблицы *Spa_management*:

-- Создание базы данных

```
CREATE DATABASE spa_management;

-- Выбор базы данных
USE spa_management;

-- Таблица пользователей
CREATE TABLE Users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    role ENUM('Guest', 'Manager', 'Staff') NOT NULL
);

-- Таблица процедур
CREATE TABLE Procedures (
    procedure_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);

-- Таблица бронирований
CREATE TABLE Bookings (
    id INT AUTO_INCREMENT PRIMARY KEY,
    procedure_id INT NOT NULL,
    guest_id INT NOT NULL,
    booking_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    status ENUM('Pending', 'Confirmed', 'Cancelled', 'Completed') NOT NULL,
    FOREIGN KEY (procedure_id) REFERENCES Procedures(procedure_id),
    FOREIGN KEY (guest_id) REFERENCES Users(id)
);
```

Целью выполнения данного курсового проекта было практическое применение полученных знаний в области объектно-ориентированного программирования, а не разработка долгосрочного программного обеспечения.

Поэтому некоторые заготовки для дальнейшего развития системы не были реализованы, что оставляет пространство для будущих улучшений. Аналогично, меньшее внимание было уделено безопасности обработки и хранения данных, что является важным аспектом для реальных приложений, но в рамках данного проекта не стало приоритетом.

В системе представлены несколько ключевых классов. Класс *User* служит абстрактной основой для всех пользователей системы, обеспечивая общие атрибуты и методы. Класс *Guest* наследует от *User* и предоставляет функционал, позволяющий взаимодействовать с системой, включая методы для бронирования процедур, отмены броней и просмотра доступных процедур. Класс *Manager* управляет процедурами и пользователями, а класс *Staff*, производный от *User*, отвечает за выполнение услуг и просмотр бронирований, что позволяет эффективно организовать рабочий процесс.

Класс *Procedure* содержит информацию о процедурах, предлагаемых в СПА-комплексе, включая название, цену и описание. Класс *Booking* хранит данные о записях клиентов на процедуры, включая дату, время и статус бронирования. Все эти классы образуют основную структуру системы, обеспечивая необходимую функциональность для управления СПА-комплексом и предоставляя пользователям удобный интерфейс для взаимодействия с системой.

Кроме того, была разработана диаграмма активности, которая иллюстрирует основные шаги взаимодействия пользователей с системой. Она помогает визуализировать процесс, делая его более понятным для разработчиков и пользователей, что в свою очередь способствует улучшению интерфейса и общей структуры программы. В целом, проект стал отличной возможностью для применения теоретических знаний на практике и разработки системы, которая может быть доработана и усовершенствована в будущем.

Диаграмма начинается с входа в систему, где пользователю предлагается выбрать свою роль: гость, управляющий или персонал. Если пользователь является гостем, он может просмотреть доступные процедуры. Если процедуры доступны, ему предоставляется возможность выбрать процедуру и указать дату и время. Далее система проверяет доступность выбранного времени. В случае успешного бронирования пользователь получает уведомление о подтверждении бронирования и может выбрать другую процедуру или завершить процесс. Если пользователь является управляющим, он имеет доступ к управлению процедурами и пользователями. Он может добавлять или удалять процедуры и пользователей, а также получать подтверждения об успешных действиях. Персонал системы также имеет возможность просматривать

бронирования и управлять процедурами. Они могут добавлять новые процедуры или отменять существующие. Диаграмма активности позволяет наглядно представить последовательность действий пользователей в системе и их взаимодействие с различными функциями. Код диаграммы активности для системы управления СПА-комплесом:

```
@startuml
start

:Войти в систему;

if (Гость?) then (да)
   :Просмотреть доступные процедуры;
   if (Процедуры доступны?) then (да)
      :Выбрать процедуру;
      :Выбрать дату и время;
      repeat
         if (Доступность на выбранное время?) then (да)
            :Подтвердить бронирование;
            :Получить уведомление о бронировании;
            if (Хотите записаться на другую процедуру?) then (да)
               :Вернуться к выбору процедуры;
               repeat
                  :Просмотреть доступные процедуры;
                  :Выбрать процедуру;
               repeat while (Процедуры доступны?) is (да)
            endif
         else (нет)
            :Выбрать другое время;
         endif
      repeat while (Доступность на выбранное время?) is (нет)
   else (нет)
      :Получить уведомление о недоступности процедур;
   endif

else (нет)
   if (Управляющий?) then (да)
      :Управлять процедурами;
      if (Добавить новую процедуру?) then (да)
```

```
    :Ввести данные процедуры;
    :Сохранить процедуру;
    :Получить подтверждение о добавлении процедуры;
  else (нет)
    if (Удалить процедуру?) then (да)
      :Выбрать процедуру для удаления;
      :Подтвердить удаление процедуры;
      :Получить уведомление об удалении;
    endif
  endif
  :Управление пользователями;
  if (Добавить нового пользователя?) then (да)
    :Ввести данные пользователя;
    :Сохранить пользователя;
    :Получить подтверждение о добавлении пользователя;
  endif
  if (Удалить пользователя?) then (да)
    :Выбрать пользователя для удаления;
    :Подтвердить удаление пользователя;
    :Получить уведомление об удалении;
  endif
else (Персонал?) then (да)
  :Просмотреть бронирования;
  if (Отменить бронь?) then (да)
    :Выбрать бронь для отмены;
    :Подтвердить отмену брони;
    :Получить уведомление об отмене брони;
  endif
  :Управлять процедурами;
  if (Добавить новую процедуру?) then (да)
    :Ввести данные процедуры;
    :Сохранить процедуру;
    :Получить подтверждение о добавлении процедуры;
  else (нет)
    if (Удалить процедуру?) then (да)
      :Выбрать процедуру для удаления;
      :Подтвердить удаление процедуры;
      :Получить уведомление об удалении;
```

```
            endif
        endif
    endif
endif

:Выйти из системы;

stop
@enduml
```

Эта диаграмма активности помогает визуализировать процесс взаимодействия пользователей с системой управления СПА-комплексом, облегчая понимание логики работы программы.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Сразу после запуска программы система предлагает пользователю выбрать под какой ролью ему зайти (см. рисунок 4.1) или вообще выйти из системы (по вводе Exit в любой части программы программа завершаеся).

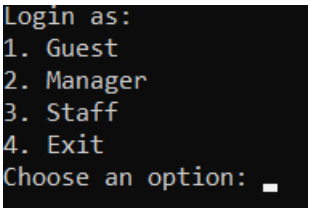

Рисунок 4.1 – Выбор роли

После выбора роли пользователю необходимо ввести почту и пароль, которые будут соответствовать выбранной роли и будут коррентными. После корректного ввода всех данных система открывает пользователю меню соответствующей роли, для примера сначала выберем *Гостя* (см. рисунок 4.2).

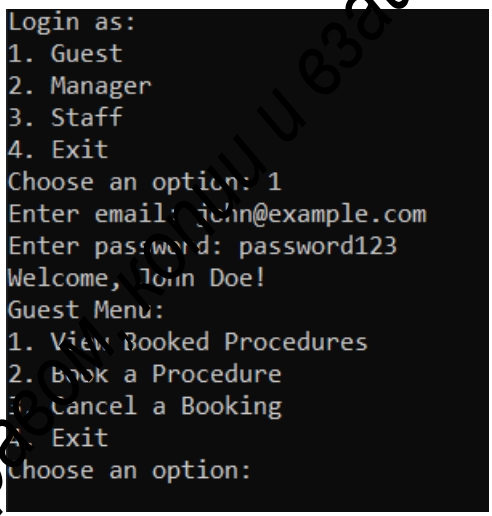

Рисунок 4.2 – Правильно введенная почта, пароль и открытие меню *Гостя*

Если пользователь неверно выбрал логин, но пароль и почта правильные, то система выдает следующее (см. рисунок 4.3).

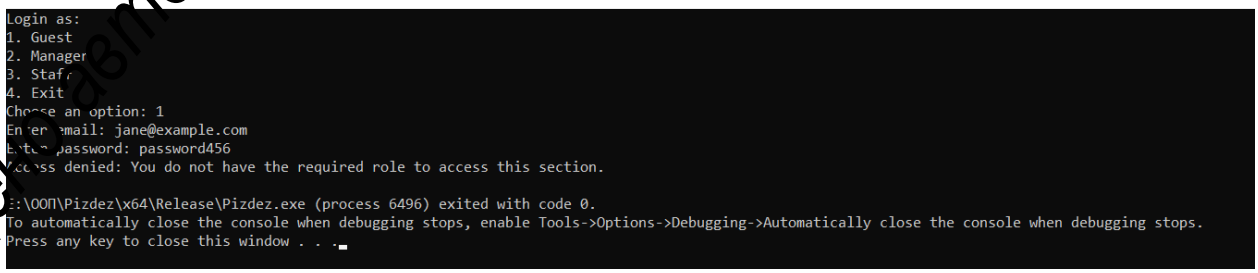

Рисунок 4.3 – Неверный выбор роли

Если при входе в систему неверно введена почта, то система покажет следующее (см. рисунок 4.4).

```
Login as:
1. Guest
2. Manager
3. Staff
4. Exit
Choose an option: 2
Enter email: шгнпекуцы
Invalid email format. Please try again:
```

Рисунок 4.4 – Почта не прошла валидацию

Если пользователь неверно ввел логин и пароль, то система покажет следующее (см. рисунок 4.5).

```
Login as:
1. Guest
2. Manager
3. Staff
4. Exit
Choose an option: 2
Enter email: шгнпекуцы
Invalid email format. Please try again: invalidmail@inv.alid
Enter password: tegsvsj
Password must be at least 8 characters. Please try again: 87wjsnbdxhkal
Incorrect email or password.

E:\ООП\Pizdez\x64\Release\Pizdez.exe (process 22404) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Рисунок 4.5 – Неверный логин и пароль

После успешного входа в систему (в данном случае под ролью Гостя) и выбора в меню Посмотреть процедуры, система покажет следующее (см. рисунок 4.6).

```
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option: 1
Your Booked Procedures:
ID| Name           | Description                                    | Price
-------------------------------------------------------------------------------
1 | Facial Treatment| A relaxing facial to rejuvenate your skin.| 50.000000
2 | Manicure       | Nail care and polish.                          | 30.000000
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option:
```

Рисунок 4.6 – Посмотреть процедуры

Если в меню пользователь выбрал Забронировать процедуру, то система покажет пользователю следующее (см. рисунок 4.7).

```
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option: 2
Available Procedures:
ID| Name            | Description                                  | Price
-------------------------------------------------------------------------
1 | Facial Treatment| A relaxing facial to rejuvenate your skin.   | 50.000000
2 | Massage Therapy | A full-body massage to relieve stress.       | 75.000000
3 | Manicure        | Nail care and polish.                        | 30.000000
Enter procedure ID to book: 3
Booking confirmed!
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option:
```

Рисунок 4.7 – Забронировать процедуру

Если пользователь решит отменить процедуру, то система отменит ее после введения пользователем *ID* той процедуры, которую он хочет отменить, и затем отменит процедуру (см. рисунок 4.8).

```
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option: 3
Your Bookings:
ID| Procedure       | Booking Date
-----------------------------------------
1 | Facial Treatment| 2024-12-19 23:46:25
4 | Manicure        | 2024-12-20 00:12:51
5 | Manicure        | 2024-12-20 03:13:54
Enter booking ID to cancel: 4
Booking cancelled successfully.
Guest Menu:
1. View Booked Procedures
2. Book a Procedure
3. Cancel a Booking
4. Exit
Choose an option:
```

Рисунок 4.7 – Отменить процедуру

Если пользователь захочет выйти из системы (полностью или чтобы зайти под другой ролью) и нажмет Exit, то система ему покажет следующее (см. рисунок 4.8).

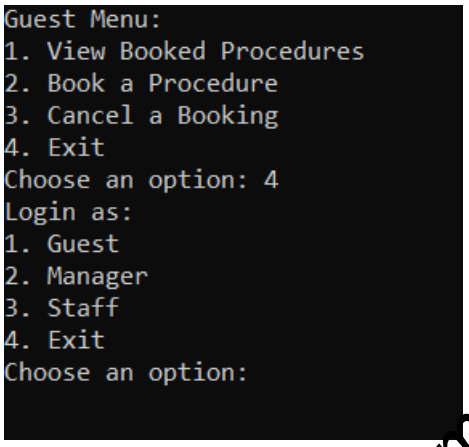

Рисунок 4.8 – Выход из профиля

Далее пользователь может или завершить работу программы или войти под другой ролью.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта был проведен глубокий анализ выбранной предметной области, на основе которого были сформулированы требования к информационной системе и определены задачи. В соответствии с техническим заданием была разработана иерархия классов для информационной системы СПА-комплекса.

В процессе изучения предметной области сделаны важные выводы о значимости СПА-комплексов в жизни современного человека. В условиях стремительного развития технологий и программного обеспечения сложно представить эффективное функционирование СПА без интегрированной информационной системы, что подчеркивает необходимость её разработки.

В ходе исследования и сравнения систем-аналогов были рассмотрены две информационные системы, предоставляющие услуги в области СПА. Анализ функционала этих систем и их возможностей позволил доработать и уточнить требования к разрабатываемой системе.

Для разработки информационной системы использовался язык программирования *C++*, поддерживающий объектно-ориентированный подход. Была создана иерархия классов, где базовым абстрактным классом стал *User*, от которого наследуются классы *Guest*, *Manager* и *Staff*. Каждый пользователь системы обладает уникальными функциями, позволяющими эффективно взаимодействовать с ней, в то время как работники СПА, особенно администраторы, могут управлять системой.

В процессе реализации программы были задействованы все четыре основных принципа объектно-ориентированного программирования (ООП), среди которых ключевое место занимали наследование и полиморфизм, реализованные с использованием виртуальных функций. Также для проектирования системы были разработаны диаграммы, наглядно иллюстрирующие структуру классов и взаимодействие пользователей с системой. Диаграммы активности демонстрируют последовательность действий пользователей, что помогает лучше понять логику работы программы.

Кроме того, для работы с базой данных была подключена *MySQL*, что обеспечило эффективное управление данными о пользователях, процедурах и записях. Использование *SQL*-запросов гарантировало надежное взаимодействие между приложением и базой данных.

Поставленные в начале проекта цели были успешно достигнуты. Реализованная система соответствует установленным требованиям и, при определенных улучшениях, может быть использована в будущем. Учитывая растущую популярность СПА-комплексов и их информационных систем, данное решение имеет все шансы на успешное применение.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] СТП 01-2017. Стандарт предприятия. Дипломные проекты (работы). Общие требования. - Минск: БГУИР, 2017. - 169 с.

[2] Навроцкий, А. А. Дипломное проектирование по специальности «Автоматизированные системы обработки информации» : учеб.-метод. пособие / А. А. Навроцкий, Н. В. Батин. – Минск : БГУИР, 2018. – 66 с.

[3] Лафоре, Р. объектно-ориентированное программирование в *C++*. – «Питер», 2004. – 928 с.

[4] MySQL Documentation [Электронный ресурс]. — Режим доступа: https://dev.mysql.com/doc/.

[6] Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд./Пер. с англ. — М.: «Издательство «Бином», СПб: «Невский диалект», 1998 г. – 560 с.

[7] PlantUML. PlantUML Web Server. [Электронный ресурс]. — Режим доступа: https://www.plantuml.com/.

[8] Страуструп, Б. Программирование: Принципы и практика использования C++ / Б. Страуструп. – «Вильямс», 2018. – 1328 с.

[9] Навроцкий, А. Основы алгоритмизации и программирования в среде *VISUAL C++* / А. Навроцкий. – Минск: БГУИР, 2014. – 160 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Программный код

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <memory>
#include <regex>
#include <mysql_driver.h>
#include <mysql_connection.h>
#include <cppconn/prepared_statement.h>
#include <cppconn/resultset.h>
#include <cstdio>
#include <algorithm>

// Refactored printTable function using printf
void printTable(const std::vector<std::string>& headers, const std::vector<std::vector<std::string>>& rows) {
    // Determine the maximum width for each column
    std::vector<size_t> columnWidths(headers.size(), 0);

    // Calculate maximum width for headers
    for (size_t i = 0; i < headers.size(); ++i) {
        columnWidths[i] = headers[i].size();
    }

    // Calculate maximum width for each column based on data rows
    for (const auto& row : rows) {
        for (size_t i = 0; i < row.size(); ++i) {
            columnWidths[i] = std::max(columnWidths[i], row[i].size());
        }
    }

    // Print headers with separators
    for (size_t i = 0; i < headers.size(); ++i) {
        printf("%-*s", static_cast<int>(columnWidths[i]), headers[i].c_str());
```

```
            if (i < headers.size() - 1) {
                printf("| "); // Separator between headers
            }
        }
        printf("\n");

        // Print separator between headers and data
        for (size_t width : columnWidths) {
            printf("%-*s", static_cast<int>(width), std::string(width, '-').c_str());
            printf("--"); // Separator for the columns
        }
        printf("\n");

        // Print data with separators
        for (const auto& row : rows) {
            for (size_t i = 0; i < row.size(); ++i) {
                printf("%-*s", static_cast<int>(columnWidths[i]), row[i].c_str());
                if (i < row.size() - 1) {
                    printf("| "); // Separator between cells
                }
            }
            printf("\n");
        }
    }

    // Abstract Base Class: User
    class User {
    protected:
        std::string name;
        std::string email;
        std::string password;
        std::string role;

        bool validateEmail(const std::string& email) {
            std::regex emailRegex(R"((\w+)(\.\w+)*@(\w+\.)(\w+))");
            return std::regex_match(email, emailRegex);
        }
```

```
    bool validatePassword(const std::string& password) {
        return password.length() >= 8;
    }

public:
    virtual ~User() = default;


    void login(std::shared_ptr<sql::Connection> con, const std::string& ex-
pectedRole) {
        std::cout << "Enter email: ";
        std::cin >> email;
        while (!validateEmail(email)) {
            std::cout << "Invalid email format. Please try again: ";
            std::cin >> email;
        }

        std::cout << "Enter password: ";
        std::cin >> password;
        while (!validatePassword(password)) {
            std::cout << "Password must be at least 8 characters. Please try
again: ";
            std::cin >> password;
        }

        try {
            std::unique_ptr<sql::PreparedStatement> pstmt(
                con->prepareStatement("SELECT name, role FROM Users
WHERE email = ? AND password = ?"));
            pstmt->setString(1, email);
            pstmt->setString(2, password);
            std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

            if (res->next()) {
                name = res->getString("name");
                role = res->getString("role");

                if (role != expectedRole) {
```

```
                std::cout << "Access denied: You do not have the required role
to access this section.\n";
                exit(0);
            }

            std::cout << "Welcome, " << name << "!\n";
        }
        else {
            std::cout << "Incorrect email or password.\n";
            exit(0);
        }
    }
    catch (sql::SQLException& e) {
        std::cerr << "SQL error: " << e.what() << std::endl;
        exit(1);
    }
}

virtual void menu(std::shared_ptr<sql::Connection> con) = 0;
};

// Utility: Print table headers
void printTableHeaders(const std::vector<std::string>& headers) {
    for (const auto& header : headers) {
        std::cout << std::setw(15) << std::left << header;
    }
    std::cout << "\n";
    std::cout << std::string(15 * headers.size(), '-') << "\n";
}

// Class: Guest
class Guest : public User {
public:
    void viewProcedures(std::shared_ptr<sql::Connection> con) {
        try {
            std::unique_ptr<sql::PreparedStatement> pstmt(con->prepareState-
ment(
```

```
            "SELECT id, name, description, price FROM Procedures WHERE
id IN (SELECT procedure_id FROM Bookings WHERE guest_email = ?)"));
            pstmt->setString(1, email);
            std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

            std::vector<std::string> headers = { "ID", "Name", "Description",
"Price" };
            std::vector<std::vector<std::string>> rows;

            while (res->next()) {
               rows.push_back({
                  std::to_string(res->getInt("id")),
                  res->getString("name"),
                  res->getString("description"),
                  std::to_string(res->getDouble("price"))
                  });
               }

            printf("Your Booked Procedures:\n");
            printTable(headers, rows);
         }
         catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
         }
      }

      void bookProcedure(std::shared_ptr<sql::Connection> con) {
         try {
            // Получаем доступные процедуры
            std::unique_ptr<sql::PreparedStatement> pstmt(con->prepareState-
ment(
               "SELECT id, name, description, price FROM Procedures"));
            std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

            // Заголовки для таблицы
            std::vector<std::string> headers = { "ID", "Name", "Description",
"Price" };
            std::vector<std::vector<std::string>> rows;
```

```
            // Считываем процедуры из базы данных
            while (res->next()) {
                rows.push_back({
                    std::to_string(res->getInt("id")),
                    res->getString("name"),
                    res->getString("description"),
                    std::to_string(res->getDouble("price"))
                    });
            }

            // Выводим доступные процедуры
            printf("Available Procedures:\n");
            printTable(headers, rows);

            // Запрашиваем ID процедуры для бронирования
            printf("Enter procedure ID to book: ");
            int procedureId;
            std::cin >> procedureId;

            // Проверяем, введен ли корректный ID
            if (std::find_if(rows.begin(), rows.end(), [&procedureId](const
std::vector<std::string>& row) {
                return std::stoi(row[0]) == procedureId;
                }) == rows.end()) {
                printf("Invalid procedure ID. Please try again.\n");
                return;
            }

            // Вставляем бронирование в базу данных
            std::unique_ptr<sql::PreparedStatement> bookStmt(con->pre-
pareStatement(
                "INSERT INTO Bookings (procedure_id, guest_email, book-
ing_date, status) VALUES (?, ?, NOW(), 'Pending')"));
            bookStmt->setInt(1, procedureId);
            bookStmt->setString(2, email);
            bookStmt->executeUpdate();
```

```
            printf("Booking confirmed!\n");
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
    }

    void manageBookings(std::shared_ptr<sql::Connection> con)
        try {
            // Запрашиваем все бронирования с состоянием "Pending" для
дальнейшей обработки
            std::unique_ptr<sql::PreparedStatement> pstmt(con->prepareState-
ment(
                "SELECT b.id, p.name, b.booking_date, b.status FROM Bookings
b "
                "JOIN Procedures p ON b.procedure_id = p.id WHERE b.status =
'Pending'"));
            std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

            // Заголовки для вывода таблицы бронирований
            std::vector<std::string> headers = { "ID", "Procedure", "Booking
Date", "Status" };
            std::vector<std::vector<std::string>> rows;

            // Считываем все бронирования с состоянием "Pending"
            while (res->next()) {
                rows.push_back({
                    std::to_string(res->getInt("id")),
                    res->getString("name"),
                    res->getString("booking_date"),
                    res->getString("status")
                    });
            }

            // Выводим таблицу бронирований
            printf("Pending Bookings:\n");
            printTable(headers, rows);
```

```
            // Запрашиваем ID бронирования для управления
            printf("Enter booking ID to approve or cancel (0 to exit): ");
            int bookingId;
            std::cin >> bookingId;

            if (bookingId == 0) {
                return;
            }

            // Спрашиваем, что делать с выбранным бронированием
            printf("Approve or Cancel this booking?\n1. Approve\n2. Cancel\nChoose an option: ");
            int option;
            std::cin >> option;

            if (option == 1) {
                // Подтверждаем бронирование
                std::unique_ptr<sql::PreparedStatement> approveStmt(con->prepareStatement(
                    "UPDATE Bookings SET status = 'Approved' WHERE id = ?"));
                approveStmt->setInt(1, bookingId);
                approveStmt->executeUpdate();

                printf("Booking ID %d has been approved.\n", bookingId);
            }
            else if (option == 2) {
                // Отменяем бронирование
                std::unique_ptr<sql::PreparedStatement> cancelStmt(con->prepareStatement(
                    "UPDATE Bookings SET status = 'Cancelled' WHERE id = ?"));

                cancelStmt->setInt(1, bookingId);
                cancelStmt->executeUpdate();

                printf("Booking ID %d has been cancelled.\n", bookingId);
            }
            else {
```

```
                printf("Invalid option.\n");
            }
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
    }


    void cancelBooking(std::shared_ptr<sql::Connection> con) {
        try {
            std::unique_ptr<sql::PreparedStatement> pstmt(con->prepareState-
ment(
                "SELECT b.id, p.name, b.booking_date FROM Bookings b JOIN
Procedures p ON b.procedure_id = p.id WHERE b.guest_email = ?"));
            pstmt->setString(1, email);
            std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

            std::vector<std::string> headers = { "ID", "Procedure", "Booking
Date" };
            std::vector<std::vector<std::string>> rows;
            std::vector<int> bookingIds;

            while (res->next()) {
                int id = res->getInt("id");
                bookingIds.push_back(id);
                rows.push_back({
                    std::to_string(id),
                    res->getString("name"),
                    res->getString("booking_date")
                    });
            }

            printf("Your Bookings:\n");
            printTable(headers, rows);

            printf("Enter booking ID to cancel: ");
            int bookingId;
```

```
            std::cin >> bookingId;

            if (std::find(bookingIds.begin(), bookingIds.end(), bookingId) ==
bookingIds.end()) {
                printf("Invalid booking ID.\n");
                return;
            }

            std::unique_ptr<sql::PreparedStatement> cancelStmt(con->pre-
pareStatement(
                "DELETE FROM Bookings WHERE id = ?"));
            cancelStmt->setInt(1, bookingId);
            cancelStmt->executeUpdate();

            printf("Booking cancelled successfully.\n");
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
    }

    void menu(std::shared_ptr<sql::Connection> con) override {
        while (true) {
            printf("Guest Menu:\n1. View Booked Procedures\n2. Book a Proce-
dure\n3. Cancel a Booking\n4. Exit\nChoose an option: ");
            std::string choice;
            std::cin >> choice;

            if(choice == "Exit") {
                exit(0);
            }

            int option = std::stoi(choice);
            switch (option) {
            case 1:
                viewProcedures(con);
                break;
            case 2:
```

```
                bookProcedure(con);
                break;
            case 3:
                cancelBooking(con);
                break;
            case 4:
                return;
            default:
                printf("Invalid choice. Please try again.\n");
            }
        }
    }
};


// Class: Manager
class Manager : public User {
public:
    void manageUsers(std::shared_ptr<sql::Connection> con) {
        while (true) {
            std::cout << "User Management:\n1. Add User\n2. View Users\n3.
Remove User\n4. Back\nChoose an option: ";
            std::string choice;
            std::cin >> choice;

            if (choice == "Exit") {
                exit(0);
            }

            int option = std::stoi(choice);
            switch (option) {
            case 1: {
                std::cout << "Enter user name: ";
                std::string userName;
                std::cin.ignore();
                std::getline(std::cin, userName);

                std::cout << "Enter user email: ";
```

```
                std::string userEmail;
                std::getline(std::cin, userEmail);

                std::cout << "Enter user password: ";
                std::string userPassword;
                std::getline(std::cin, userPassword);

                std::cout << "Enter user role (Guest/Manager/Staff): ";
                std::string userRole;
                std::getline(std::cin, userRole);

                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "INSERT INTO Users (name, email, password, role) VAL-
UES (?, ?, ?, ?)"));
                    pstmt->setString(1, userName);
                    pstmt->setString(2, userEmail);
                    pstmt->setString(3, userPassword);
                    pstmt->setString(4, userRole);
                    pstmt->executeUpdate();

                    std::cout << "User added successfully!\n";
                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;

            case 2: {
                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "SELECT id, name, email, role FROM Users"));
                    std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

                    std::cout << "Users:\n";
```

```
                std::vector<std::string> headers = { "ID", "Name", "Email",
"Role" };
                std::vector<std::vector<std::string>> rows;

                while (res->next()) {
                    rows.push_back({
                        std::to_string(res->getInt("id")),
                        res->getString("name"),
                        res->getString("email"),
                        res->getString("role")
                        });
                }

                printTable(headers, rows); // Using the printTable function here
            }
            catch (sql::SQLException& e) {
                std::cerr << "SQL error: " << e.what() << std::endl;
            }
            break;
        }
        case 3: {
            std::cout << "Enter user ID to remove: ";
            int userId;
            std::cin >> userId;

            try {
                std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                    "DELETE FROM Users WHERE id = ?"));
                pstmt->setInt(1, userId);
                pstmt->executeUpdate();

                std::cout << "User removed successfully!\n";
            }
            catch (sql::SQLException& e) {
                std::cerr << "SQL error: " << e.what() << std::endl;
            }
            break;
```

```
            }
            case 4:
                return;
            default:
                std::cout << "Invalid choice. Please try again.\n";
            }
        }
    }

    void manageProcedures(std::shared_ptr<sql::Connection> con) {
        while (true) {
            std::cout << "Procedure Management:\n1. View Procedures\n2. Add
Procedure\n3. Remove Procedure\n4. Back\nChoose an option: ";
            std::string choice;
            std::cin >> choice;

            if (choice == "Exit") {
                exit(0);
            }

            int option = std::stoi(choice);
            switch (option) {
            case 1: {
                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "SELECT id, name, description, price FROM Procedures"));
                    std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

                    std::cout << "Procedures:\n";
                    std::vector<std::string> headers = { "ID", "Name", "Descrip-
tion", "Price" };
                    std::vector<std::vector<std::string>> rows;

                    while (res->next()) {
                        rows.push_back({
                            std::to_string(res->getInt("id")),
                            res->getString("name"),
```

```
                res->getString("description"),
                std::to_string(res->getDouble("price"))
                });
            }

            printTable(headers, rows); // Using the printTable function here
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
        break;
    }
    case 2: {
        std::cout << "Enter procedure name: ";
        std::string procedureName;
        std::cin.ignore();
        std::getline(std::cin, procedureName);

        std::cout << "Enter procedure description: ";
        std::string procedureDescription;
        std::getline(std::cin, procedureDescription);

        std::cout << "Enter procedure price: ";
        float procedurePrice;
        std::cin >> procedurePrice;

        try {
            std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                "INSERT INTO Procedures (name, description, price) VAL-
UES (?, ?, ?)"));
            pstmt->setString(1, procedureName);
            pstmt->setString(2, procedureDescription);
            pstmt->setDouble(3, procedurePrice);
            pstmt->executeUpdate();

            std::cout << "Procedure added successfully!\n";
        }
```

```
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }
            case 3: {
                std::cout << "Enter procedure ID to remove: ";
                int procedureId;
                std::cin >> procedureId;

                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "DELETE FROM Procedures WHERE id = ?"));
                    pstmt->setInt(1, procedureId);
                    pstmt->executeUpdate();

                    std::cout << "Procedure removed successfully!\n";
                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }
            case 4:
                return;
            default:
                std::cout << "Invalid choice. Please try again.\n";
        }
    }

    void menu(std::shared_ptr<sql::Connection> con) override {
        while (true) {
            std::cout << "Manager Menu:\n1. Manage Users\n2. Manage Pro-
cedures\n3. Exit\nChoose an option: ";
            std::string choice;
            std::cin >> choice;
```

```
            if (choice == "Exit") {
                exit(0);
            }

            int option = std::stoi(choice);
            switch (option) {
            case 1:
                manageUsers(con);
                break;
            case 2:
                manageProcedures(con);
                break;
            case 3:
                return;
            default:
                std::cout << "Invalid choice. Please try again.\n";
            }
        }
    }
};


// Class: Staff
class Staff : public User {
public:


    void manageBookings(std::shared_ptr<sql::Connection> con) {
        while (true) {
            std::cout << "Booking Management:\n"
                << "1. View Bookings\n"
                << "2. Cancel Booking\n"
                << "3. Change Booking Status\n"
                << "4. Confirm Booking\n"
                << "5. Back\n"
                << "Choose an option: ";
```

```
        std::string choice;
        std::cin >> choice;

        int option;
        try {
            option = std::stoi(choice);
        }
        catch (std::exception& e) {
            std::cout << "Invalid input. Please enter a number.\n";
            continue;
        }

        switch (option) {
        case 1: {
            // View Bookings
            try {
                std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                    "SELECT id, user_id, procedure_id, booking_date, status
FROM Bookings"));
                std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

                std::cout << "Bookings:\n";
                std::vector<std::string> headers = { "ID", "User ID", "Proce-
dure ID", "Booking Date", "Status" };
                std::vector<std::vector<std::string>> rows;

                while (res->next()) {
                    rows.push_back({
                        std::to_string(res->getInt("id")),
                        std::to_string(res->getInt("user_id")),
                        std::to_string(res->getInt("procedure_id")),
                        res->getString("booking_date"),
                        res->getString("status")
                        });
                }

                if (rows.empty()) {
```

```
                std::cout << "No bookings found.\n";
            }
            else {
                printTable(headers, rows); // Function to print the table
            }
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
        break;
    }

    case 2: {
        // Cancel Booking
        std::cout << "Enter booking ID to cancel: ";
        int bookingId;
        std::cin >> bookingId;

        try {
            std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                "UPDATE Bookings SET status = 'Canceled' WHERE id =
?"));
            pstmt->setInt(1, bookingId);
            int affectedRows = pstmt->executeUpdate();

            if (affectedRows > 0) {
                std::cout << "Booking canceled successfully!\n";
            }
            else {
                std::cout << "Booking not found or already canceled.\n";
            }
        }
        catch (sql::SQLException& e) {
            std::cerr << "SQL error: " << e.what() << std::endl;
        }
        break;
    }
```

```
            case 3: {
                // Change Booking Status
                std::cout << "Enter booking ID to change status: ";
                int bookingId;
                std::cin >> bookingId;

                std::cout << "Enter new status (Pending/Confirmed/Canceled): ";
                std::string newStatus;
                std::cin >> newStatus;

                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "UPDATE Bookings SET status = ? WHERE id = ?"));
                    pstmt->setString(1, newStatus);
                    pstmt->setInt(2, bookingId);
                    int affectedRows = pstmt->executeUpdate();

                    if (affectedRows > 0) {
                        std::cout << "Booking status updated to " << newStatus <<
"!\n";
                    }
                    else {
                        std::cout << "Booking not found.\n";
                    }
                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }

            case 4: {
                // Confirm Booking
                std::cout << "Enter booking ID to confirm: ";
                int bookingId;
                std::cin >> bookingId;
```

```
                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "UPDATE Bookings SET status = 'Confirmed' WHERE id =
?"));
                    pstmt->setInt(1, bookingId);
                    int affectedRows = pstmt->executeUpdate();

                    if (affectedRows > 0) {
                        std::cout << "Booking confirmed successfully!\n";
                    }
                    else {
                        std::cout << "Booking not found or already confirmed.\n";
                    }
                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }

            case 5:
                return; // Exit the function

            default:
                std::cout << "Invalid choice. Please try again.\n";
            }
        }
    }


    void manageProcedures(std::shared_ptr<sql::Connection> con) {
        while (true) {
            std::cout << "Procedure Management:\n1. View Procedures\n2. Add
Procedure\n3. Remove Procedure\n4. Back\nChoose an option: ";
            std::string choice;
            std::cin >> choice;
```

```
            int option = std::stoi(choice);
            switch (option) {
            case 1: {
                // Просмотр процедур
                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "SELECT id, name, description, price FROM Procedures"));
                    std::unique_ptr<sql::ResultSet> res(pstmt->executeQuery());

                    std::cout << "Procedures:\n";
                    std::vector<std::string> headers = {"ID", "Name", "Descrip-
tion", "Price" };
                    std::vector<std::vector<std::string>> rows;

                    while (res->next()) {
                        rows.push_back({
                            std::to_string(res->getInt("id")),
                            res->getString("name"),
                            res->getString("description"),
                            std::to_string(res->getDouble("price"))
                            });
                    }

                    printTable(headers, rows); // Использование функции для
вывода таблицы

                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }
            case 2: {
                // Добавление новой процедуры
                std::cout << "Enter procedure name: ";
                std::string procedureName;
                std::cin.ignore();
```

```
                std::getline(std::cin, procedureName);

                std::cout << "Enter procedure description: ";
                std::string procedureDescription;
                std::getline(std::cin, procedureDescription);

                std::cout << "Enter procedure price: ";
                float procedurePrice;
                std::cin >> procedurePrice;

                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "INSERT INTO Procedures (name, description, price) VAL-
UES (?, ?, ?)"));
                    pstmt->setString(1, procedureName);
                    pstmt->setString(2, procedureDescription);
                    pstmt->setDouble(3, procedurePrice);
                    pstmt->executeUpdate();

                    std::cout << "Procedure added successfully!\n";
                }
                catch (sql::SQLException& e) {
                    std::cerr << "SQL error: " << e.what() << std::endl;
                }
                break;
            }
            case 3: {
                // Удаление процедуры
                std::cout << "Enter procedure ID to remove: ";
                int procedureId;
                std::cin >> procedureId;

                try {
                    std::unique_ptr<sql::PreparedStatement> pstmt(con->pre-
pareStatement(
                        "DELETE FROM Procedures WHERE id = ?"));
                    pstmt->setInt(1, procedureId);
```

```
                pstmt->executeUpdate();

                std::cout << "Procedure removed successfully!\n";
            }
            catch (sql::SQLException& e) {
                std::cerr << "SQL error: " << e.what() << std::endl;
            }
            break;
        }
        case 4:
            return;  // Возврат в основное меню
        default:
            std::cout << "Invalid choice. Please try again.\n";
        }
    }
}


void menu(std::shared_ptr<sql::Connection> con) override {
    while (true) {
        std::cout << "Staff Menu: \n1. Manage Bookings\n2. Manage Proce-
dures\n3. Exit\nChoose an option: ";
        std::string choice;
        std::cin >> choice;

        int option = std::stoi(choice);
        switch (option) {
        case 1:
            manageBookings(con);
            break;
        case 2:
            manageProcedures(con);  // Опция для управления процедурами
            break;
        case 3:
            return;  // Выход из меню
        default:
            std::cout << "Invalid choice. Please try again.\n";
        }
```

```
        }
    }
};

int main() {
    std::shared_ptr<sql::Connection> con;
    try {
        sql::mysql::MySQL_Driver* driver = sql::mysql::get_mysql_driver_in-
stance();
        con.reset(driver->connect("tcp://127.0.0.1:3306", "root", "root"));
        con->setSchema("spa_management");

        while (true) {
            std::cout << "Login as:\n1. Guest\n2. Manager\n3. Staff\n4.
Exit\nChoose an option: ";
            std::string choice;
            std::cin >> choice;

            if (choice == "4" || choice == "Exit") {
                break;
            }

            int option = std::stoi(choice);
            switch (option) {
            case 1: {
                auto guest = std::make_shared<Guest>();
                guest->login(con, "Guest");
                guest->menu(con);
                break;
            }
            case 2: {
                auto manager = std::make_shared<Manager>();
                manager->login(con, "Manager");
                manager->menu(con);
                break;
            }
            case 3: {
                auto staff = std::make_shared<Staff>();
```

```
                staff->login(con, "Staff");
                staff->menu(con);
                break;
            }
            case 4: {
                break;
            }
            default:
                std::cout << "Invalid choice. Please try again.\n";
            }
        }
    }
    catch (sql::SQLException& e) {
        std::cerr << "Connection failed: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

ВЕДОМОСТЬ КУРСОВОЙ РАБОТЫ

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КР 6-05-0612-03 158 ПЗ	Пояснительная записка	59 с.
	Графические документы	

Изм.	Лист	№ докум.	Подп	Дата	БГУИР КП 6-05-0612-03 158 Д1		
Разраб.		Хаджинова			Система управления СПА-комплексом. Ведомость курсового проекта.	Лист 59	Листов 59
Провер.		Ючков				Кафедра ИТАС гр. 320604	