Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Программирование на языках
высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему
«Планировщик занятий»

Студент Н.С. Горчаков

Руководитель Е.В. Богдан

МИНСК 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ

(подпись)
2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту: *Горчакову Никите Сергеевичу*
1. Тема проекта *«Планировщик занятий»*
2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*
3. Исходные данные к проекту Программа написана в помощь преподавателям и студентам в выполнении их повседневной деятельности
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
1. Введение.
2. Задание.
3. Обзор литературы.
3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
4.1. Структура входных и выходных данных.
4.2. Разработка диаграммы классов.
4.3. Описание классов.
5. Разработка программных модулей.
5.1. Разработка схем алгоритмов (два наиболее важных метода).
5.2. Разработка алгоритмов (описание алгоритмов по шагам для двух методов).
6. Результаты работы.
7. Заключение
8. Литература
9. Приложения
5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Диаграмма классов.
2. Схема алгоритма `DBType* Add (WriteType* element).`
3. Схема алгоритма `DBType* GetFromDB (bsoncxx::oid oid).`
6. Консультант по проекту (с обозначением разделов проекта) Е.В.Богдан
7. Дата выдачи задания *15 сентября 2023 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;
разделы 2, 3 – 10 %;
разделы 4 к –20 %;
разделы 5 к – 35 %;
раздел 6,7,8 – 5 %;
раздел 9 к – 5%;
оформление пояснительной записки и графического материала к 15.12.23 – 10 %
Защита курсового проекта с 21.12 по 28.12.23г.
РУКОВОДИТЕЛЬ Е.В.Богдан

Задание принял к исполнению _____ Н.С.Горчаков
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В 2023 году общество уже не может представить свою жизнь без информационных технологий (далее ИТ). Информационные технологии облегчают его жизнь, и увеличивают эффективность труда. Одной из областей жизни, которая может быть усовершенствована с помощью ИТ это система образования.

Целью данного курсового проекта является разработка «Планировщик занятий». Эта система предназначена для автоматизации процессов сбора, обработки и хранения информации о занятиях, группах преподавателях и их расписании. Она позволяет упростить и ускорить работу преподавателях, повысить точность и актуальность расписаний, что является важной частью студенческой жизни.

В рамках данного проекта были разработаны следующие функциональные возможности:

1. Регистрация различных типов пользователей (преподаватель, студент, администратор);
2. Просмотр и редактирование информации о преподавателях, группах, студентах и их расписании;
3. Добавление новых студентов и элементов расписания;

Проект выполнен с использованием фреймворка Qt и языка программирования C++, что обеспечивает высокую производительность. Для хранения информации была выбрана нереляционная СУБД MongoDB и соответствующий базе данных драйвер mongocxx.

В дальнейшем планируется расширение функциональности системы, добавление новых инструментов для анализа данных, а также интеграция с другими информационными системами университета создание и создание собственного кластера данных.

В заключении хотелось бы отметить, что разработка данной системы является актуальной задачей, способной повысить эффективность работы военкоматов и качество обслуживания призывников.

В итоге мы имеем проект, который призван облегчить жизнь как студентов, так и администрации университета, а при дальнейшей разработке и увеличении функционала может распространиться на другие типы учебных заведений как школы и колледжи.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор методов и алгоритмов решения поставленной задачи

1.1.1 Объектно-ориентированное программирование

Объектно-ориентированное программирование – это методология программирования, которая использует объекты для моделирования и организации кода. В C++ ООП реализуется с помощью классов и объектов [7].

Классы в C++ являются шаблонами для создания объектов. Они определяют, какие данные и функции будут содержать объекты этого класса.

Объекты в C++ – это экземпляры класса. Каждый объект имеет свой собственный набор данных (атрибутов) и функций (методов), определенных в его классе.

ООП в C++ поддерживает четыре основных принципа:

1. Инкапсуляция: Инкапсуляция означает сокрытие деталей реализации и объединение данных и методов в одном объекте. В приложении, например, данные пользователя инкапсулированы в объектах класса userData;
2. Наследование: Наследование позволяет создавать новые классы на основе существующих, переиспользуя их код и добавляя новые функции. В приложении все окна наследуются от базового класса Form;
3. Полиморфизм: Полиморфизм позволяет использовать один и тот же интерфейс для различных типов данных;
4. Абстракция: Абстракция означает представление сложных систем через более простые интерфейсы.

1.1.2 Базы данных MongoDB

MongoDB представляет наиболее популярную на данный момент систему управления базами данных. По разным оценкам входит в десяток самых используемых баз данных в мире.

Ключевой особенностью это системы является её документа-ориентированность. Если стандартные базы данных хранят строки, то MongoDB хранит документы. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

Приложение также использует mongocxx драйвер для работы с базой данных. драйвер представляет библиотеку, написанную на языке C++, и которая реализует систему подключения к удалённому серверу, на котором фактически храниться весь набор данные.

Драйвер реализует свою кроссплатформенность по средством Source-only поставки. Source-only поставка перекладывает обязательство компиляции

под каждую систему на разработчика приложения, что позволяет облегчить реализацию кроссплатформенности для разработчика библиотеки.

СУБД MongoDB использует для передачи данных BSON. BSON – это сокращение от binary JSON. BSON позволяет работать с данными быстрее: быстрее выполняется поиск и обработка. Хотя надо отметить, что BSON в отличие от хранения данных в формате JSON имеет небольшой недостаток: в целом данные в JSON-формате занимают меньше места, чем в формате BSON, с другой стороны, данный недостаток с лихвой окупается скоростью[9].

1.1.3 Фреймворк QT

Для создания графического интерфейса также используется Qt. Qt — мощный фреймворк для разработки кроссплатформенных приложений, который предоставляет разработчикам широкие возможности для создания графических пользовательских интерфейсов (GUI) и реализации функционала программ. Qt – полностью объектно-ориентированная библиотека. Новая концепция “Signals and slots” являются ключевой концепцией в фреймворке Qt и представляют собой механизм, который обеспечивает эффективное взаимодействие между объектами в Qt-приложениях. Эта система связывает события (signals) с обработчиками (slots), обеспечивая гибкость и модульность кода. Также имеется возможность обработки событий, например нажатия клавиш клавиатуры, нажатия элементов интерфейса.

1.2 Разработка требований к функционалу

«Планировщик занятий» – приложение, которое представляет собой систему управления базой данных, которая позволяет пользователям просматривать, добавлять, редактировать и удалять записи. Функционал может зависеть от типа пользователя:

1. Просмотр записей: пользователи типа администратор, студент могут просматривать все записи в базе данных. Записи отображаются в таблице, где каждая строка представляет собой отдельную запись;
2. Добавление записей: пользователи типа администратор могут добавлять новые записи в базу данных. Для этого они должны заполнить форму с информацией о новой записи;
3. Редактирование записей: пользователи типа администратор могут редактировать существующие записи в базе данных. Для этого они должны выбрать запись для редактирования и изменить информацию в форме редактирования;
4. Удаление записей: пользователи типа администратор могут удалять существующие записи из базы данных. Для этого они должны выбрать запись для удаления и подтвердить свое действие;

5. Авторизация пользователей: приложение поддерживает систему авторизации пользователей типа администратор, студент. Пользователи должны ввести свои учетные данные для входа в систему;

6. Регистрация пользователей: новые пользователи типа студент могут зарегистрироваться в системе, создав новую учетную запись;

7. Реализация кэша загруженных объектов для уменьшения общего времени отклика приложения.

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

2.1 Структура входных и выходных данных

Приложение работает с двумя основными типами входных данных:

1. Данные пользователя: это данные, которые пользователь вводит через GUI. Они включают в себя информацию для регистрации нового пользователя, данные для входа в систему существующего пользователя, а также данные, которые пользователь хочет добавить, редактировать или удалить в базе данных;

2. Базы данных: это данные, которые приложение получает из базы данных MongoDB. Они включают в себя информацию о призывниках и зарегистрированных, которые хранятся в базе данных.

Приложение генерирует следующие выходные данные:

1. Данные GUI: это данные, которые отображаются пользователю через GUI. Они могут включать в себя сообщения об ошибках, подтверждения успешных действий и информацию из базы данных;

2. Данные базы данных: это данные, которые приложение записывает в базу данных MongoDB. Они могут включать в себя информацию о новых пользователях и обновленные данные существующих пользователей;

3. Сигналы: приложение также генерирует сигналы как часть механизма сигналов и слотов Qt. Сигналы используются для обработки событий, таких как нажатие кнопок пользователем;

2.2 Разработка диаграммы классов

Диаграмма классов – это структурная диаграмма языка моделирования UML, которая демонстрирует общую структуру иерархии классов системы, их кооперацій, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними. Она широко применяется не только для документирования и визуализации, но также для конструирования посредством прямого или обратного проектирования.

На диаграмме классы представлены в рамках, содержащих три компонента:

1. В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом;

2. В средней части перечислены атрибуты (поля) класса;

3. В нижней части перечислены методы класса.

Целью создания диаграммы классов является графическое представление статической структуры классов.

Диаграмма классов для приложения «Планировщик занятий» приведена в приложении А.

2.3 Описание классов

2.3.1 Класс StartWindow

Класс `StartWindow` является начальным окном, которое загружает `StudentWindow`, `AdminWindow`. А также по совместительству окном авторизации.

Конструктор (`StartWindow::StartWindow (QWidget *parent)`) : это конструктор класса `Form`. Он принимает указатель на `QWidget`, представляющий родительский виджет и настраивает пользовательский интерфейс для окна.

Связка поля State а также его гетра `StatesEnum GetState();` и сетера `void SetState(StatesEnum state)` обеспечивают превращение кнопки login в signin и наоборот а также изменения её функционала.

Слот `SwitchState()` является обработчиком события нажатия кнопки смены поведения окна `StartWindow`. Пример на рисунке 2.3.1.1, и рисунке 2.3.2.2.

Слот `LogIn()` является обработчиком события нажатия на кнопку login/Signin

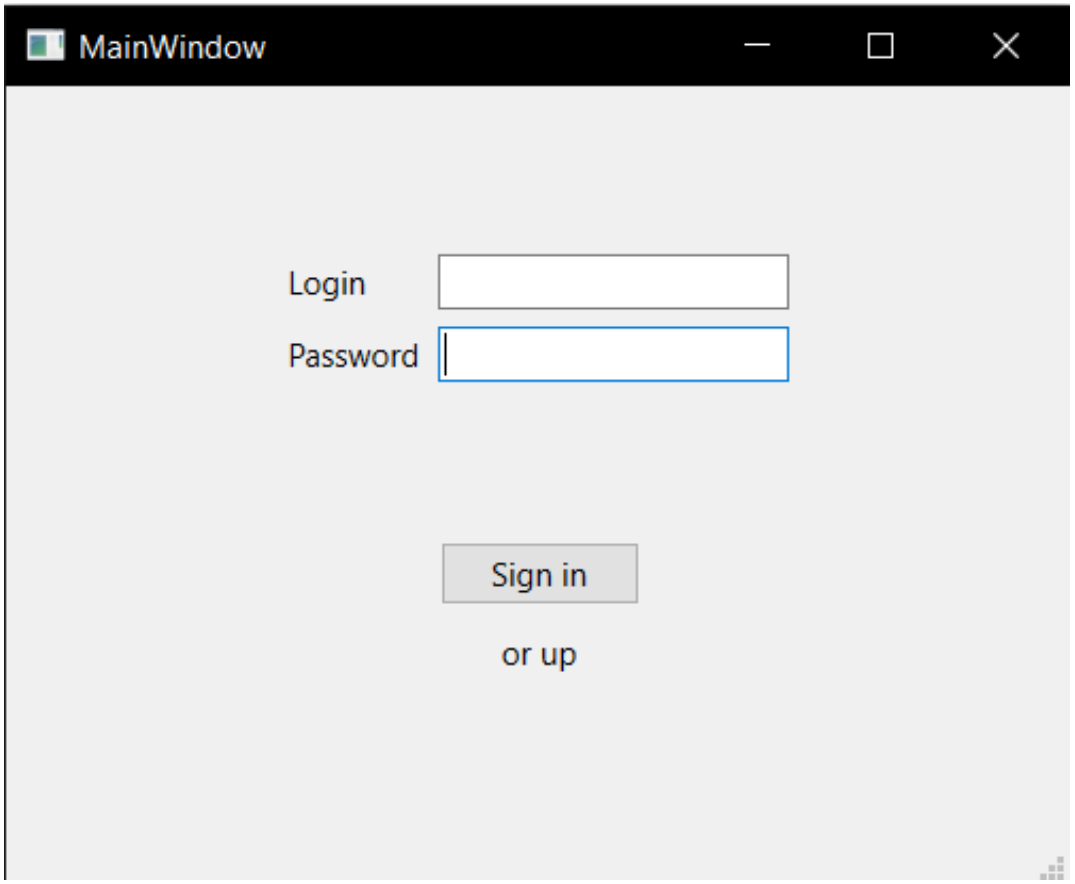

Рисунок 2.3.1.1 – Демонстрация окна `StartWindow`.

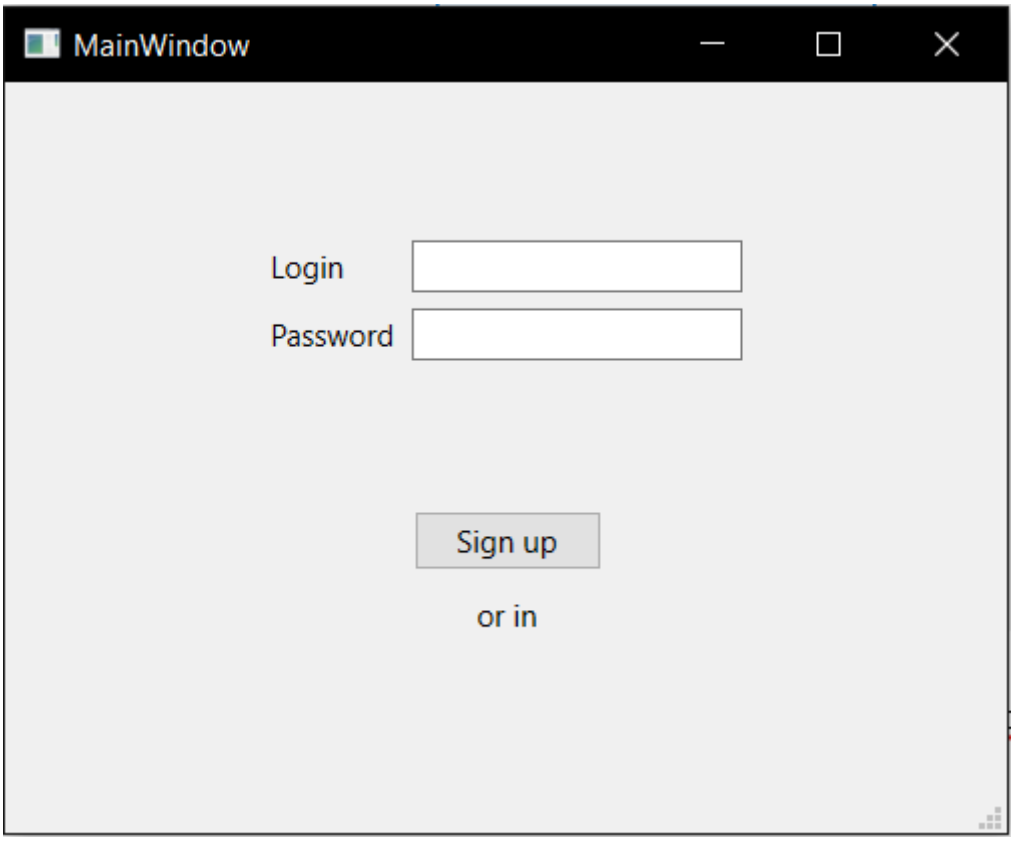

Рисунок 2.3.1.2 – Демонстрация окна `StartWindow` во втором состоянии.

2.3.2 Класс StudentWindow

Класс `StudentWindow` является главным окном взаимодействия студента и приложения, в котором он может просмотреть данные своего профиля и расписание его группы.

Метод `void setAccAndFill(AccountDB* Acc)` вызывается после того как с базы данных было получено подтверждение наличия учётной записи с соответствующими логином и паролем. Сам метод выполняет функции заполнения окна данными и обновления таблицы в центре экрана.

Слот void Setdate(QDate date) является обработчиком события изменения даты в календаре, и вызывает метод UpdateTable объекта Schedule класса ScheduleTable для обновления его содержимого.

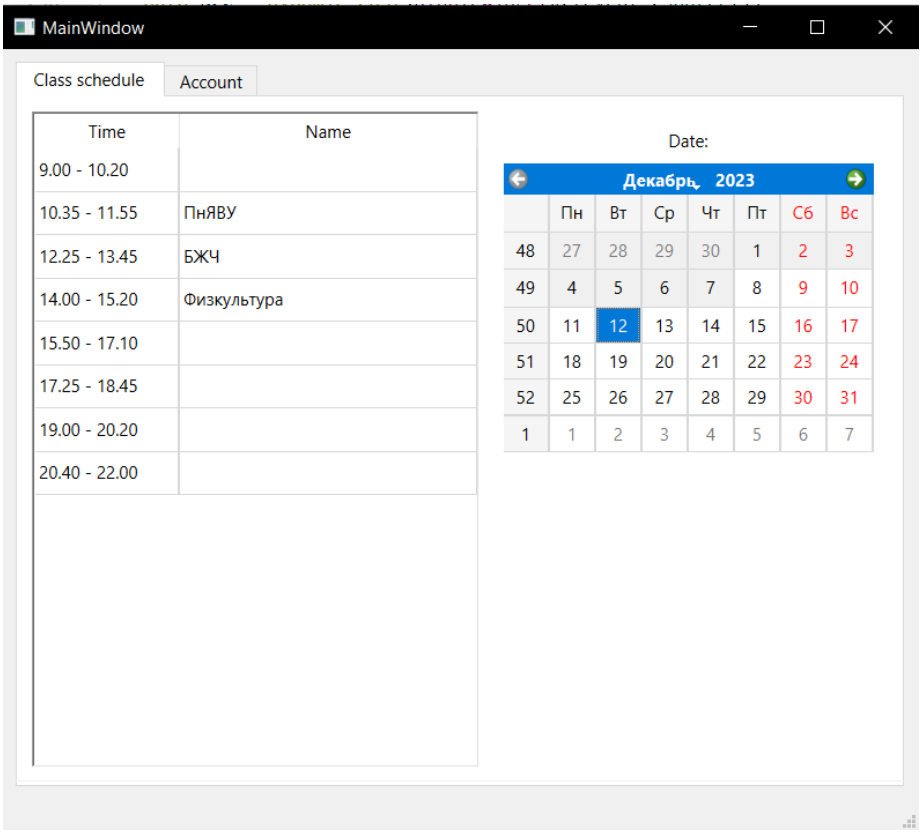

Рисунок 2.3.2 – Демонстрация окна `StudentWindow`

2.3.3 Класс AdminWindow

Класс `AdminWindow` является главным окном управления дынными находящимися в базе данных. Представляет собой три вкладки каждая из которых отведена под свой тип данных.

Событие `resizeEvent(QResizeEvent *event)` override переопределяет событие `resizeEvent` базового класса для сохранения взвешенной ширины окна.

Слот `ChangeCurentSize(int tab)` вместе с полем `initialWidth` выполняют функцию изменения длины окна при переключении вкладок.

Вкладка “Accounts” рисунок 2.3.3.1 на ней представлена система управления пользовательскими учётными записями. Для удобства пользователя здесь размещена таблица класса `AccountTable` и виджет класса `DisplayAccountWidget` для упрощения просмотра и редактирования данных.

Слоты `AddNewAccount()`, `UpdateAccount()`, `DeleteAccount()` выполняют функции удаления, обновления и добавления учётных записей пользователей также вызываются при нажатии на соответствующие кнопки в интерфейсе.

Слот `SetAccountToDisplayWidget(AccountDB* acc)` помещает сведения о пользователе для демонстрации дополнительной информации в `DisplayAccountWidget`.

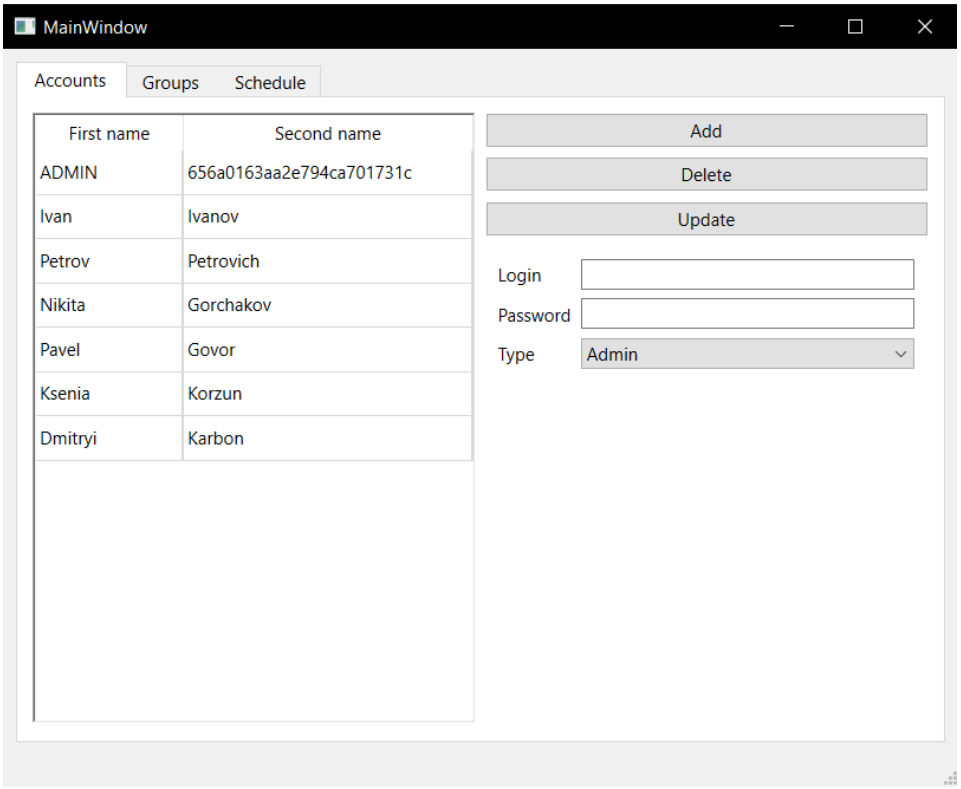

рисунок 2.3.3.1 – Демонстрация вкладки Accounts
окна `MainWindow`

Вкладка “Groups” рисунок 2.3.3.2 на ней представлена система управления группами студентов. Для удобства пользователя здесь были размешены таблицы классов `GroupsDBTable`, `GroupedAccountsTable` и `UngroupedAccountsTable` для упрощения просмотра и редактирования данных. Также на этой вкладке реализовано взаимодействие «drag and drop» между таблицами `GroupedAccountsTable и UngroupedAccountsTable`.

Метод `UpdateGroupsScrean(GroupDB* grp = NULL)` обновляет все таблицы во вкладке.

Слоты `AddNewGroup()`, `DeleteGroup()`, `UpdateGroup()` выполняют функции удаления, обновления и добавления информации о группах также вызываются при нажатии на соответствующие кнопки в интерфейсе.

Слот `SetGroup(GroupDB* acc)` вызывается при выборе элементы из списка групп при наличии такого и обновляет данные в таблице класса `GroupedAccountsTable`.

Слот `AddAccToGroup(AccountDB* acc)` вызываемый при перетаскивание записи об аккаунте из списка не сгруппированных записей учётных данных пользователя в список сгруппированных записей учётных данных пользователя, обновляет данные группы добавляя туда пользователя.

Слот `DeleteAccFromGroup(AccountDB* acc)` вызываемый при перетаскивание записи об аккаунте из списка сгруппированных записей

учётных данных пользователя в список не сгруппированных записей учётных данных пользователя, обновляет данные группы убирая от туда пользователя.

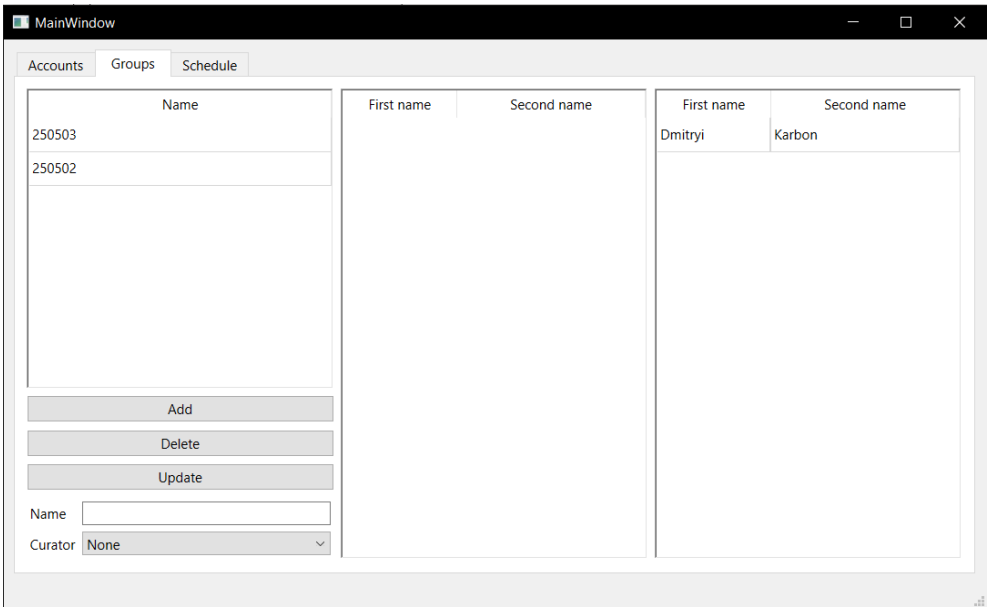

рисунок 2.3.3.2 – Демонстрация вкладки Groups
окна `MainWindow`

Вкладка “Schedule” рисунок 2.3.3.3 на ней представлена система управления расписанием групп студентов. Для удобства пользователя здесь были размешены таблицы классов `GroupsDBTable`, `TeachersTable`, `ScheduleTable` и `PossibleLessonsTable` для упрощения просмотра и редактирования данных. Также на этой вкладке реализовано взаимодействие «drag and drop» между таблицами `ScheduleTable` и `PossibleLessonsTable`.

Метод `UpdateScheduleScrean(AccountDB* teacher = NULL, GroupDB* grp = NULL)` обновляет все таблицы во вкладке.

Слоты `AddClass()` и `DeleteClass()` выполняют функции удаления и добавления информации о типах занятий которые могут проводить преподаватели, также вызываются при нажатии на соответствующие кнопки в интерфейсе.

Слот `DataChanged(QDate date)` вызывается при изменении выбранной даты в календаре, и обновляет данные во таблице `ScheduleTable`.

Слот `SetGroupSch(GroupDB* acc)` вызывается при изменении выбранной записи в таблицу `GroupsDBTable`, и записывает данные в поле group.

Слот `AddSchedule(std::string str, int RowIndex)` вызываемый при перетаскивание информации о уроке из списка занятий которые может провести преподаватель в список занятий запанированных на дату выбранную в календаре, добавляет запись в расписание.

Слот `DeleteSchedule(ScheduleDB* acc)` вызываемый при перетаскивание записи об аккаунте из список занятий запанированных на дату выбранную в календаре в списка занятий которые может провести преподаватель, убирает запись из расписания.

2.3.4 Класс DBController

Класс `DBController`, это класс, отвечающий за все действия с базой данных: подключение обновление чтение. Реализует паттерн `Singletone` который предотвращает повторное создание объекта этого класса. Содержит в себе 5 классов отвечающие за манипуляции связанные с коллекциями соответствующих им типов.

Конструктор `DBController()` поочерёдно инициализирует поля хранящие объекты подклассов после чего проверяет подключение к интернету.

Поле `DBController*` singleton и его метод `DBController* GetInstance()` являются реализацией паттерна singleton, также с этой целью копирующий конструктор и оператор приравнивания созданы приватно.

2.3.5 Класс Controller

Шаблонный класс Controller является базовым классом, определяющим общее поведение для его наследников.

Поле `std::list<DBType*> Cash` представляет собой временное хранилище, прочитанных из базы данных, записей что позволяет уменьшить задержки срабатывания клавиш.

Метод bool `IsAlredyInCash(bsoncxx::oid oid, DBType **res)` проверяет находиться ли объект в Cash.

Метод `DBType* GetFromDB(bsoncxx::oid oid)` принимает в себя `id` записи для последующего считывания и возвращает проинициализированный объект класса соответствующего документу.

Метод `int GetCount()` возвращает число записанных записей.

Метод `std::list<DBType*>GetAll()` возвращает лист всех документов записанных в базу данных. Для оптимизации работы этого метода существует поле `bool IsChanged` в случае активности которого не происходит считывания, а просто возвращается копия Cash.

Методы `DBType* Add(WriteType* element)`, `bool Delete(bsoncxx::oid oid)` и `bool Update(DBType* el)` отвечают за добавление, удаление и обновление записей соответственно.

Конструктор `Controller(mongocxx::collection collection)` принимает в себя уже открытую коллекцию для инициализации.

2.3.6 Класс AccountsController

Класс `AccountsController` является специализацией шаблонного класса Controller для классов `AccountDB` и `Account`.

Метод `std::list<AccountDB*>GetAllTeachers()` возвращает все записи учётных данных педагогического персонала.

Метод bool `IsAccountExist(LogPass passLog)` проверяет комбинацию пароля и логина на присутствие в базе данных. Требуется для работы авторизации.

Метод `AccountDB* FindFullAccount(LogPass passLog)` считывает из базы данных а после возвращает инициализированный объект соответствующий данной паре логина и пароля.

2.3.7 Класс GroupsController

Класс `GroupsController` является специализацией шаблонного класса Controller для классов `GroupDB` и `GroupDB`.

2.3.8 Класс ScheduleController

Класс `ScheduleController` является специализацией шаблонного класса Controller для классов `ScheduleDB` и `Schedule`.

Метод `std::list<ScheduleDB*>GetAllSchedulesinDate(int JulianDate, GroupDB* group)` возвращает всё расписание для группы в указанный день.

2.3.9 Класс InfosController

Класс `InfosController` является специализацией шаблонного класса Controller для классов `AdditionalInfoDB` и `AdditionalInfo`.

Метод `AdditionalInfoDB* GetFromDB(bsoncxx::oid oid, AccountType type)` переписывает метод базового класса подразумевая то, что в коллекции записей дополнительной информации может храниться два типа документов: `TeacherInfoDB` и `StudentInfoDB`.

Метод `std::list<AdditionalInfoDB*> GetAll()` переписывает метод базового класса подразумевая то, что в коллекции записей дополнительной информации может храниться два типа документов: `TeacherInfoDB` и `StudentInfoDB`.

`AdditionalInfoDB* Add(AdditionalInfo* accInf, AccountType type)` переписывает метод базового класса подразумевая то, что в коллекции записей дополнительной информации может храниться два типа документов: `TeacherInfoDB` и `StudentInfoDB`. В дополнение принимает тип аккаунта, который будет записан.

2.3.10 Интерфейс ImyQTableWidget

Интерфейс `ImyQTableWidget` заставляет все наследуемые от него классы реализовывать метод `UpdateTable()`.

2.3.11 Класс AccountsTable

Класс `AccountsTable` наследован от класса `QTableWidget` и служит для отображения элементов коллекции записей учётных данных пользователей. Реализует интерфейс `ImyQTableWidget` с методом `UpdateTable()`.

Конструктор `AccountsTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<AccountDB*> getAccounts()` получает все записи учётных данных пользователей, и возвращает их список.

Слот `ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn)` служит для обработки события изменения выбранной строки и испускает сигнал `AccountDBChanged(AccountDB* acc)` с соответствующими данными об учётной записи пользователя.

2.3.12 Класс UngroupedAccountsTable

Класс `UngroupedAccountsTable` наследован от класса `AccountsTable` и служит для отображения элементов коллекции записей учётных данных пользователей, которые не состоят в какой-либо группе.

Конструктор `UngroupedAccountsTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<AccountDB*> getAccounts()` получает агрегированные для этой таблицы записи учётных данных пользователей, и возвращает их список.

Метод `dropEvent(QDropEvent *event)` переопределяет базовый обработчик события для системы «drag and drop» который в свою очередь испускает сигнал `DelAcc(AccountDB* acc)` с соответствующими данными для удаления.

2.3.13 Класс GroupedAccountsTable

Класс `GroupedAccountsTable` наследован от класса `AccountsTable` и служит для отображения элементов коллекции записей учётных данных пользователей, которые находятся в группе.

Конструктор `GroupedAccountsTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<AccountDB*> getAccounts()` получает агрегированные для этой таблицы записи учётных данных пользователей, и возвращает их список.

Метод `dropEvent(QDropEvent *event)` переопределяет базовый обработчик события для системы «drag and drop» который в свою очередь испускает сигнал `AddAcc (AccountDB* acc)` с соответствующими данными для добавления.

Метод `setGroup(GroupDB* grp)` является сеттером для поля `group` которое учувствует в агрегации записей учётных данных пользователей.

2.3.14 Класс TeachersTable

Класс `TeachersTable` наследован от класса `AccountsTable` и служит для отображения элементов коллекции записей учётных данных членов педагогического коллектива. Реализует интерфейс `ImyQTableWidget` с методом `UpdateTable()`.

Конструктор `TeachersTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<AccountDB*> getAccounts()` получает агрегированные для этой таблицы записи учётных данных членов педагогического коллектива, и возвращает их список.

2.3.15 Класс GroupsDBTable

Класс `GroupsDBTable` наследован от класса `QTableWidget` и служит для отображения элементов коллекции записей групп. Реализует интерфейс `ImyQTableWidget` с методом `UpdateTable()`.

Конструктор `GroupsDBTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<GroupDB*> getGroups()` получает все записи групп студентов, и возвращает их список.

Слот `ActiveRowChanged(int currentRow, int currentColumn, int previousRow, int previousColumn)` служит для обработки события изменения выбранной строки и испускает сигнал `AccountDBChanged(GroupDB* acc)` с соответствующими данными об группе студентов.

2.3.16 Класс PosibleLessonsTable

Класс `PosibleLessonsTable` наследован от класса `QTableWidget` и служит для отображения элементов коллекции типов лекций, которые может провести преподаватель. Реализует интерфейс `ImyQTableWidget` с методом `UpdateTable()`.

Конструктор `PosibleLessonsTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `std::list<std::string> getLessonsTypes()` получает все записи о лекциях которые может провести преподаватель, и возвращает их список.

Метод `dropEvent(QDropEvent *event)` переопределяет базовый обработчик события для системы «drag and drop» который в свою очередь испускает сигнал `DeleteSchedule(ScheduleDB* acc)` с соответствующими данными для удаления.

Метод `setTeacher(TeacherInfoDB* teach)` является сеттером для поля `teacher` которое учувствует в агрегации возможных лекций.

2.3.17 Класс ScheduleTable

Класс `ScheduleTable` наследован от класса `QTableWidget` и служит для отображения элементов коллекции расписания, которые может провести преподаватель. Реализует интерфейс `ImyQTableWidget` с методом `UpdateTable()`.

Конструктор `ScheduleTable (QWidget *parent = 0)` вызывает базовый конструктор класса и передаёт в него родительский `QWidget`.

Метод `setSchedules(std::list<ScheduleDB*> scheds)` заполняет массив расписания.

Метод `dropEvent(QDropEvent *event)` переопределяет базовый обработчик события для системы «drag and drop» который в свою очередь испускает сигнал `AddSchedule(std::string str, int RowIndex)` с соответствующими данными для добавления.

2.3.18 Интерфейс IDocable

Интерфейс `IDocable` служит для обозначения возможности преобразования класса в документ типа Bson. Интерфейс предполагает для реализации методы: `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

Метод `document::view toDoc()` это виртуальный метод который определяет какие из виртуальных методов Build() должны быть запущены, что позволяет строить удобную сборочную линию для возвращаемого документа.

Метод `builder::stream::document* Build(builder::stream::document* builder)` принимает в себя объект

строителя из порождающего паттерна Builder. Builder — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов [10].

2.3.19 Интерфейс IFromDB

Интерфейс `IFromDB` служит для определения функционала классов объектов, которые являются репрезентацией данных полученных из базы данных. Является дополнением интерфейса `IDocable` ведь каждый объект созданный из BSON документа может быть обратно в него преобразован.

Поле doc хранит документ, из которого был сконструирован объект, соответствующий идентификатору, записанному в поле id.

Поля `int OwnedCount` и `bool isExist` следят за количество ссылок на этот объект, и предотвращают доступ к этому классу в случае его удаления.

Конструктор `IFromDB(document::value fromDoc)` принимает документ и инициализирует свои поля.

Переопределяет метод Build для реализации интерфейса `IDocable`.

2.3.20 Класс Group

Класс Group реализует интерфейс `IDocable`. Group это класс созданный для записи объекта `Group` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию.

Конструктор `Group(std::string name)` принимает имя группы для инициализации.

Поле Students хранит все ссылки на документы студентов, которые состоят в группе. Имеет сеттер `setStudents(std::list<StudentInfoDB*> students)`.

Поле `Curator` хранит ссылку на запись преподавателя, который назначен куратором для этой группы. Имеет сеттер setCurator(TeacherInfoDB* teacher).

Поле `Name` хранит название группы.

Для реализации интерфейса `IDocable` переопределяет методы `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

2.3.21 Класс LogPass

Класс `LogPass` реализует интерфейс `IDocable`. `LogPass` это класс созданный для записи комбинации пароля и логина. Служит только для нахождения уже имеющейся записи учётных данных студента.

Конструктор `LogPass(std::string login, std::string password)` принимает логин и пароль для инициализации соответствующих им переменных.

Поля `Login` и `Password` хранят комбинацию логина и пароля.

Для реализации интерфейса `IDocable` переопределяет методы `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

2.3.22 Класс Account

Класс `Account` реализует интерфейс `IDocable`. `Account` это класс созданный для записи объекта `Group` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию. Наследован от `LogPass`.

Конструктор `Account(std::string login, std::string password, AccountType accountType)` принимает логин, тип записи и пароль для инициализации соответствующих им переменных. Использует конструктор `LogPass` для инициализации переменных, предлежащих этому классу.

Поле `AccountType AccountType` хранят тип записи.

Для реализации интерфейса `IDocable` переопределяет методы `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

2.3.23 Класс AdditionalInfo

Класс `AdditionalInfo` реализует интерфейс `IDocable`. `Account` это класс созданный для записи объекта `AdditionalInfo` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию.

Поле `FirstName` хранит имя.

Поле `SecondName` хранит фамилию.

Поле `Owner` хранит ссылку на объект записи учётных данных пользователя, и имеет сеттер `setOwner(AccountDB* acc)`.

Для реализации интерфейса `IDocable` переопределяет методы `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

2.3.24 Класс Schedule

Класс `Schedule` реализует интерфейс `IDocable`. `Schedule` это класс созданный для записи объекта `Schedule` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию.

Конструктор `Schedule(QDate date, std::string name, enum Para paraNumber)` принимает дату, название и номер пары для инициализации соответствующих им переменных.

Поле `Date` хранит дату.

Поле `Name` хранит имя.

Поле `ParaNumber` хранит номер пары.

Поле `Teacher` хранит ссылку на объект записи дополнительной информации о преподователе, и имеет сеттер `setTeacher(TeacherInfoDB* teacher)`.

Поле `Group` хранит ссылку на объект группы, и имеет сеттер `setGroup(GroupDB* acc)`.

Для реализации интерфейса `IDocable` переопределяет методы `document::view toDoc()` и `builder::stream::document* Build(builder::stream::document* builder)`.

2.3.25 Класс StudentInfo

Класс `StudentInfo` реализует интерфейс `IDocable`. `StudentInfo` это класс созданный для записи объекта `StudentInfo` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию.

Конструктор `StudentInfo(std::string firstName, std::string secondName, int curs)` принимает имя, фамилию и номер курса для инициализации соответствующих им переменных.

2.3.26 Класс TeacherInfo

Класс `TeacherInfo` реализует интерфейс `IDocable`. `TeacherInfo` это класс созданный для записи объекта `TeacherInfo` в базу данных после записи проходит удаление и должен быть заменён на свою DB версию.

Конструктор `TeacherInfo(std::string fritstName, std::string secondName, std::string faculty)` принимает имя, фамилию и название факультета для инициализации соответствующих им переменных.

2.3.27 Класс AccountDB

Класс `AccountDB` реализует интерфейс `IFromDB` и наследуется от класса `Account` создан для репрезентации объекта, который был прочитан с базы данных.

2.3.28 Класс ScheduleDB

Класс `ScheduleDB` реализует интерфейс `IFromDB` и наследуется от класса `Schedule` создан для репрезентации объекта, который был прочитан с базы данных.

2.3.29 Класс GroupDB

Класс `GroupDB` реализует интерфейс `IFromDB` и наследуется от класса `Group` создан для репрезентации объекта, который был прочитан с базы данных.

2.3.30 Класс AdditionalInfoDB

Класс `AdditionalInfoDB` реализует интерфейс `IFromDB` и наследуется от класса `AdditionalInfo` создан для репрезентации объекта, который был прочитан с базы данных.

2.3.31 Класс TeacherInfoDB

Класс `TeacherInfoDB` реализует интерфейс `IFromDB` и наследуется от класса `TeacherInfo` создан для репрезентации объекта, который был прочитан с базы данных.

2.3.32 Класс StudentInfoDB

Класс `StudentInfoDB` реализует интерфейс `IFromDB` и наследуется от класса `StudentInfo` создан для репрезентации объекта, который был прочитан с базы данных.

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

3.1 Разработка схем алгоритмов

Метод `DBType* GetFromDB (bsoncxx::oid oid)` предназначен для извлечения данных из базы данных.

Схема метода `DBType* GetFromDB (bsoncxx::oid oid)` показана в приложении Б.

Метод `DBType* Add (WriteType* element)` позволяет добавлять данные в базу данных.

Схема метода `DBType* Add (WriteType* element)` показана в приложении В.

3.2 Разработка алгоритмов

3.2.1 Алгоритм поиска данных записи по идентификационному номеру.

1. Создание переменной для результата с типом возвращаемого значения.
2. Если переменная уже находится в кэше.
3. Тогда вернуть найденную переменную.
4. Иначе найти документ из базы данных, где id равен полученному.
5. Если не получено значение.
6. То вернуть ничего.
7. Создать переменную типа значения из переменной типа вида.
8. Создать объект возвращаемого типа из результата поиска.
9. Добавить ссылку на объект в кэш.
10. Увеличить счётчик владения объектом на 1.
11. Вернуть объект.

3.2.2 Алгоритм добавления данных в базу данных

1. Задание флага `IsChanged` в положение `true`.
2. Создание переменной для результата с типом возвращаемого значения.
3. Преобразование переданного объекта в BSON документ.
4. Выполнение операции вставки документа в коллекцию.
5. Освобождение памяти переданного объекта.
6. Если при операции вставки документа ничего не было получено.
7. То вернуть ничего.
8. Если количество вставленных документов не равно 1.
9. То выбросить ошибку
10. Получить id назначенное вставленному документу.
11. Найти документ соответствующий этом id.
12. Создать объект возвращаемого типа из результата поиска.
13. Добавить ссылку на объект в кэш.
14. Увеличить счётчик владения объектом на 1.
15. Вернуть объект.

УСЛОВНЫЕ СОКРАЩЕНИЯ

1. GUI – Graphical User Interface
2. СУБД – система управления базами данных
3. ООП – объектно-ориентированное программирование
4. UML – Unified Modeling Language
5. SQL – Structured Query Language

ЗАКЛЮЧЕНИЕ

В рамках данной работы было успешно разработано приложение на базе фреймворка Qt, предназначенное для управления данными о студентах, их расписании, группах и преподавателях. Приложение обладает функциональностью регистрации и авторизации пользователей, а также предоставляет возможность добавления, удаления и редактирования данных и просмотра статуса для студентов, групп, расписания и преподавателей.

Была реализована работа с базой данных MongoDB, что позволило осуществлять NoSQL-запросы для обработки данных. Также для удобства были использованы модели Qt для отображения данных, что обеспечило наглядность взаимодействия пользователя с приложением.

В ходе работы были использованы различные методы для достижения поставленной цели. Был проведен анализ требований к функционалу и характеристикам приложения, разработаны алгоритмы на языке программирования C++ для управления приложением и обработки данных.

В заключение можно отметить, что разработанное приложение может значительно облегчить жизнь преподавателей и студентов, обеспечивая им комфортную работу и удобный доступ к информации о студентах, расписании, группах и преподавателях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Рожнова, Н.Г. Вычислительные машины, системы и сети. Дипломное проектирование : учебно-метод.пособие / Н.Г. Рожнова, Н.А. Искра, И.И. Глецевич. – Минск : БГУИР, 2014. – 96 с. : ил.

[2] СТП 01–2013. Дипломные проекты (работы): общие требования. – Введ. [Электронный ресурс]. – 2013. – Режим доступа: http://www.bsuir.by/online/showpage.jsp.

[3] Единая система конструкторской документации (ЕСКД): справ. пособие / С.С. Борушек [и др.]. – М. : Изд-во стандартов, 1989. – 352 с

[4] Шлее М. - Qt4. Профессиональное программирование на C+/ Шлее М. - Л.:Наука, 2013. - 770 с.

[5] Бланшет Ж., Саммерфилд М. - QT 4: программирование GUI на C+/ Бланшет Ж., Саммерфилд М. М. — М. : ALT Linux, 2015. — 948 с. : ил.

[6] А.В. Чеботарев Библиотека Qt 4. Программирование прикладных приложений в среде Linux / А.В. Чеботарев - Л.: Наука, 2013. - 821 с.

[7] Страуструп, Б. Язык программирования C++ / Б. Страуструп. - М. : БИНОМ, 2004.- 1098 с.

[8] Дейтел, Х. Как программировать на C++ / Х. Дейтел, П. Дейтел. - М. : БИНОМ, 2001. - 1152 с.

[9] Metanit. Введение в MongoDB. [Электронный ресурс]. – Режим доступа: https://metanit.com/nosql/mongodb/1.1.php.

[10] Redactoring guru. Порождающие паттерны. [Электронный ресурс]. – Режим доступа: https://refactoring.guru/ru/design-patterns/builder.

ПРИЛОЖЕНИЕ А
(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)

Блок-схема метода

ПРИЛОЖЕНИЕ В
(обязательное)

Блок-схема метода

ПРИЛОЖЕНИЕ Г
(обязательное)

Код программы