

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 ПОСТАНОВКА ЗАДАЧИ.....	6
2 ОБЗОР ЛИТЕРАТУРЫ.....	7
2.1 Обзор методов и алгоритмов решения поставленной задачи .....	7
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ .....	10
3.1 Структура входных и выходных данных.....	10
3.2 Разработка диаграммы классов .....	10
3.3 Описание классов.....	11
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ .....	20
4.1 Разработка схем алгоритмов (два наиболее важных метода).....	20
4.2 Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).....	20
4.2.1 Алгоритм проверки повторения void checkClicked() класса RepetitionProcessWindow .....	20
4.2.2 Алгоритм инициализации колоды void initialisation() класса Deck...	20
5 РЕЗУЛЬТАТЫ РАБОТЫ.....	22
ЗАКЛЮЧЕНИЕ .....	26
ЛИТЕРАТУРА .....	27
ПРИЛОЖЕНИЕ А .....	28
ПРИЛОЖЕНИЕ Б .....	29
ПРИЛОЖЕНИЕ В .....	30
ПРИЛОЖЕНИЕ Г .....	31

## ВВЕДЕНИЕ

Целью данного курсового проекта является разработка обучающей программы с системой интервального повторения, основанной на языке программирования C++.

Интервальное повторение (англ. spaced repetition) – это методика эффективного запоминания информации, основанная на последовательности повторений с оптимальными промежутками времени между ними.

Система интервального повторения (SRS) – это методический подход, используемый для эффективного запоминания информации с помощью периодического повторения материала в определенные интервалы времени. Она основывается на принципах психологии памяти, с учетом того, что человеческий мозг лучше сохраняет информацию, которая повторяется с промежутками времени.

Существуют различные виды памяти: мгновенная, кратковременная, оперативная и долговременная. Первые три безусловно важны, но когда речь идёт о применении информации, то такого уровня запоминания зачастую не хватает для эффективного её использования.

Метод интервальных повторений напрямую влияет на нашу долговременную память. Долговременная память позволяет хранить информацию неограниченное время, именно поэтому она наиболее важна для учебы. Данный метод широко применяется в обучении и воспитании для закрепления и усвоения различных знаний или навыков.

# 1 ПОСТАНОВКА ЗАДАЧИ

Разработка программы с графическим пользовательским интерфейсом.

Программа позволит создавать наборы «карточек» с вопросами и ответами, и автоматически определять промежутки времени для их повторения, учитывая уровень запоминания пользователя.

Главные особенности программы:

1. Возможность создания и редактирования карточек с вопросами и ответами.

2. Автоматическое определение оптимального времени для повторения каждой карточки на основе алгоритмов интервального повторения.

3. Поддержка различных типов карточек, включая простые карточки для запоминания, карточки с вопросом и ответом в текстовом и графическом виде.

4. Отслеживание прогресса и результатов для обеспечения эффективности обучения

5. Понятный и простой в использовании пользовательский интерфейс, который позволит пользователям комфортно работать со сформированными карточками.

Также в программе необходимо разработать иерархию классов с использованием наследования. Для реализации программы используется объектно–ориентированный язык программирования C++.

## 2 ОБЗОР ЛИТЕРАТУРЫ

### 2.1 Обзор методов и алгоритмов решения поставленной задач

История SRS началась с исследования немецкого психолога Германа Эббингауза в конце XIX века. В своей работе "О памяти: исследования экспериментальной памяти" Эббингауз провел серию экспериментов, чтобы выяснить, как люди запоминают информацию и какое влияние на этот процесс оказывает время. Из его экспериментов произошла "кривая забывания" – график, показывающий, как быстро информация забывается после изучения. График представлен на рисунке 2.1.1.

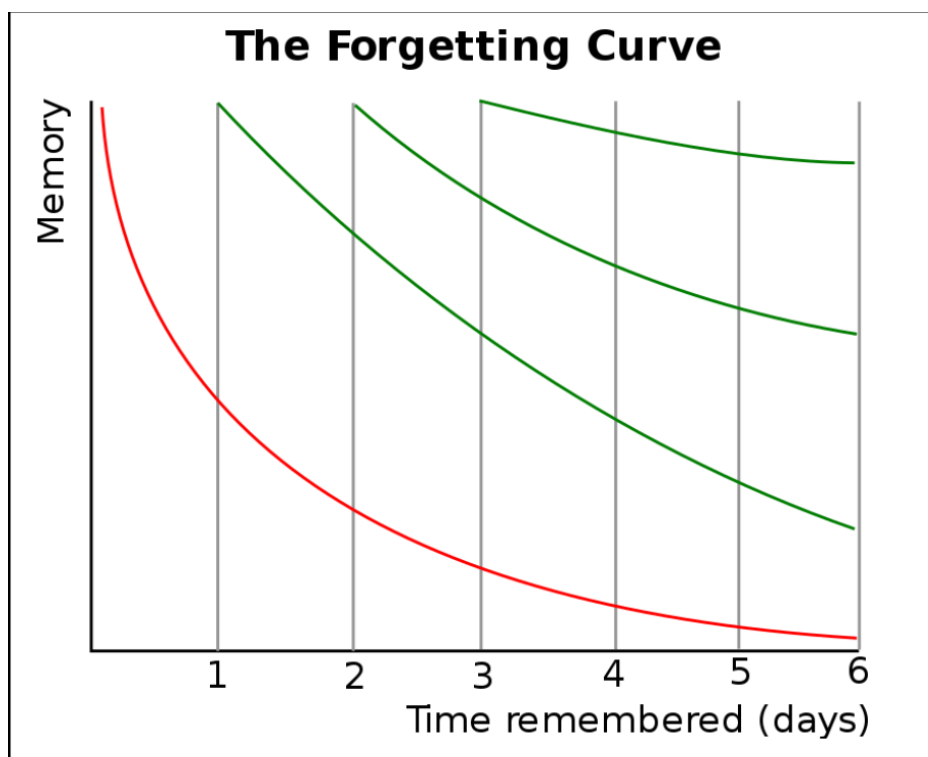


Рисунок 2.1.1 – Кривая забывания и повторения

Суть в том, что скорость забывания информации обратно пропорциональна времени с момента ее запоминания. Чем больше прошло времени, тем медленнее она запоминается.

Система интервального повторения стала основой для разработки множества приложений в разных областях. Вот несколько примеров.

1. Anki[1] – это одно из самых популярных приложений для интервального повторения. Оно позволяет создавать карточки с вопросами и ответами, которые будут периодически повторяться в зависимости от того, насколько хорошо пользователь запомнил материал. Интерфес представлен на рис. 2.1

Пользователь может создавать собственные наборы карточек или импортировать готовые наборы, доступные в библиотеке Anki. Когда пользователь изучает карточки, Anki анализирует его ответы и определяет, насколько хорошо он запомнил каждую карточку. Затем Anki автоматически регулирует интервалы повторения, чтобы пользователь повторял материал в оптимальное время, максимально увеличивая его запоминающую способность.

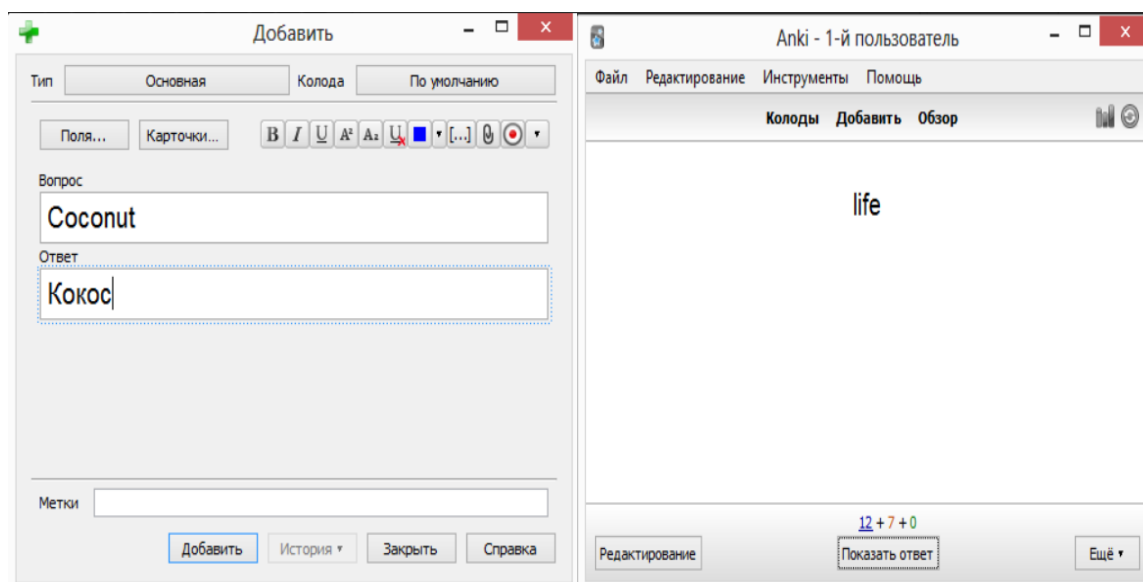


Рисунок 2.1 – Скриншоты “Anki”

2. Quizlet[2] – это приложение, в котором пользователи могут создавать наборы карточек для изучения различных тем. SRS–алгоритм автоматически определяет интервалы повторения для каждой карточки, чтобы помочь пользователю эффективно запоминать материал. Интерфейс представлен на рис. 2.2.

С помощью приложения Quizlet пользователи могут создавать наборы карточек, содержащих вопросы и ответы, по различным темам. Карточки могут быть оформлены в виде текста, изображений и даже аудио или видео материалов, что делает процесс изучения более интерактивным.

Одной из особенностей Quizlet является его обширная библиотека публичных наборов карточек, созданных другими пользователями. Это позволяет находить готовые наборы по различным предметам и темам, что удобно для самообразования или коллективного использования в рамках учебного курса.

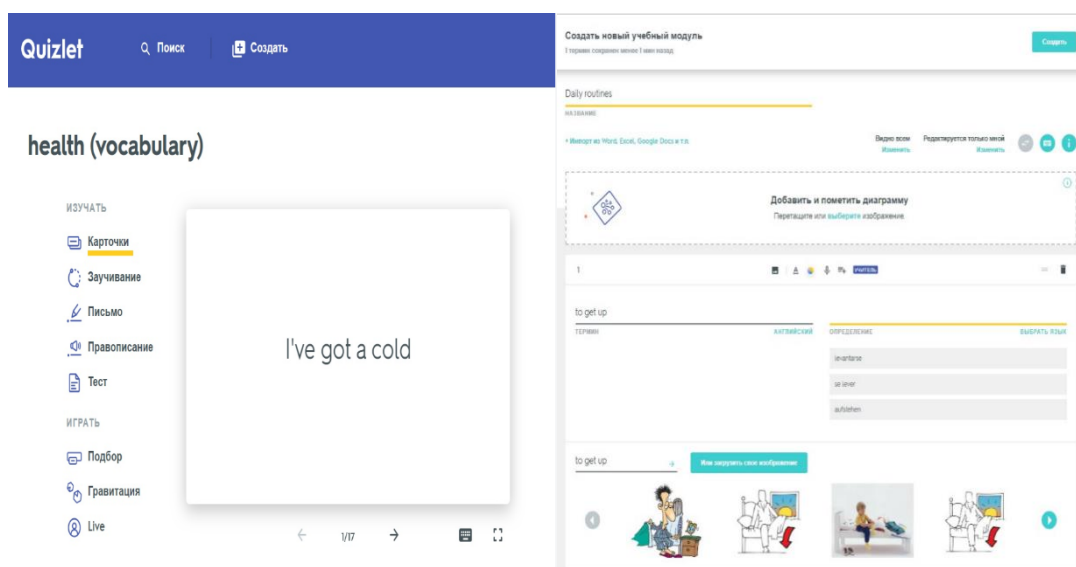


Рисунок 2.2 – Скриншоты “Quizlet”

Приложения такого типа не требуют очень тонкого визуального оформления, поскольку это не их основная задача.

Графическая библиотека Qt с её функционалом посредством сигналов (signals) и слотов (slots) сможет обеспечить возможность создания многооконного приложения, а визуальных возможностей хватит для осуществления основной цели такого рода приложений – обучение.

Для реализации алгоритма интервального повторения будет использована формула

$$Y = 2X + 1,$$

где  $Y$  — день, когда информация начнёт забываться, забывание идет постепенно и равняется  $Y$  дням,  $X$  — день последнего повторения после заучивания. Потенциал конечного интервала равен бесконечности.

## 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

### 3.1 Структура входных и выходных данных

Файл <ключ>Cards.txt (<ключ> выбирается в зависимости от темы колоды) хранит информация о каждой карточке в данной колоде в порядке их добавления пользователем. Структура файла представлена в таблице 3.1.

Таблица 3.1 – Информация о карточках в колоде

Name	Colors
Text (если есть)	ООП: инкапсуляция, наследование, полиморфизм, абстракция
Question (если есть)	Сколько всего цветов радуги?
Answer (если есть)	Семь
AddressPicture (если есть)	D://...
NextDateOfRepetition	02.11.2023
NextTimeOfRepetition	14:01
LevelOfMemorization	3
Activity	True

Пояснение – Name – название карточки, Text – текст карточки, Question – вопрос карточки, Answer – ответ на вопрос, AddressPicture – путь к картинке, NextDateOfRepetition – следующая дата повторения, NextTimeOfRepetition – следующее время повторения, LevelOfMemorization – уровень запоминания, Activity – готовность карточки с повторению.

Файл statistics.txt хранит статистику всей сессии обучения. Структура файла представлена в таблице 3.2.

Таблица 3.2 – Статистика

AmountDecks	2
AmountCards	56
AmountAnswers	154
AmountRightAnswers	150
Percentage	0.55

Пояснение – AmountDecks – количество колод, AmountCards – количество карточек, AmountAnswers – количество ответов, AmountRightAnswers – количество правильных ответов, Percentage – процент правильных ответов.

### 3.1 Разработка диаграммы классов

Диаграмма классов представлена в приложении А.

### 3.2 Описание классов

#### 1. Класс Card

Абстрактный класс, содержащий информацию о карточке.

Поля:

std::string type – тип карточки;  
std::string Name – название карточки;  
unsigned LevelOfMemorization – уровень знания по карточке;  
std::string NextTimeOfRepetition – время для повторения;  
std::string NextDateOfRepetition – дата для повторения;  
bool Activity; – готовность к повторению.

Методы:

Card(): – конструктор по умолчанию;  
virtual ~Card() – виртуальный деструктор;  
virtual void show()=0 – чисто виртуальная функция;  
std::string getName() – установка имени;  
void setName(std::string) – получение имени;  
unsigned getLevelOfMemorization() – получение уровня знаний;  
void setLevelOfMemorization(unsigned) – установка уровня знаний;  
std::string getNextTimeOfRepetition() – получение времени для повторения;  
void setNextTimeOfRepetition(std::string) – установка времени для повторения;  
std::string getNextDateOfMemorization() – получение даты для повторения;  
void setNextDateOfMemorization(std::string) – установка даты для повторения;  
bool getActivity() – получение готовности;  
void setActivity(bool) – установка готовности;  
void setType(std::string) – установить тип карточки;  
std::string getType() – получить тип карточки.



## 2. Класс Deck

Класс колоды. Содержит список карточек.

Данные:

Statistics\*statistics – указатель на объект класса статистика;  
std::string Topic – тема колоды;  
std::list<Card\*> Cards – список карточек колоды;  
std::list<Card\*> ActiveCards – список активных карточек колоды.

Методы:

explicit Deck(Statistics\*) – конструктор;  
void createCard(std::string\*,bool) – добавление карточки;  
void createFileCard(std::string\*) – добавления карточки в файл;  
void initialisation() – инициализация колоды из файлов;  
void deleteCard(const std::string&) – удаления карточки;  
void setTopic(std::string) – установка темы; 00  
std::list<Card\*> getCards() – получение списка карточек; 0  
void setActiveCards(std::list<Card\*>) – установка активных  
карточек;  
std::list<Card\*> getActiveCards() – получение активных  
карточек;  
std::string static replaceAll(std::string&, const  
std::string&,const std::string&) – замена символов в строке;  
std::string static readAnyString(std::ifstream&,char\*,  
std::string ) – считывание строки из файла;  
void rewriteInFile() – перезапись файла колоды.

## 3. Класс Session

Класс пользовательской сессии. Содержит все колоды и статистику.

Поля:

Statistics\*statistics – указатель на класс статистики;  
std::map<std::string, Deck\*> Decks – словарь колод;  
std::map<std::string, Deck\*> ActiveDecks – словарь активных  
колод.

Методы:

Session() – конструктор;  
void createDeck(const std::string&,bool) – добавление колоды;  
void createFileDeck(const std::string&) – создание файла  
колоды;  
void initialisation() – инициализация колоды из файла;  
std::map <std::string,Deck\*> getDecks() – получение колод;  
std::map <std::string,Deck\*> getActiveDecks() – получение

активных колод;

`void setActiveDecks(std::map<std::string, Deck*>) – установка`

активных колод;

`void deleteDeck(const std::string&) – удаление колоды;`

`static void deleteFileDeck(const std::string&) – удаление файла`

колоды;

`Statistics*getStatistics() – получение статистики.`

## 4. Класс Statistics

Класс статистики. Содержит все статистические данные программы.

Поля:

`unsigned AmountDecks – количество колод;`

`unsigned AmountAnswers – количество ответов;`

`unsigned AmountRightAnswers – количество верных ответов;`

`unsigned AmountCards – количество карточек;`

`float Percentage – процент правильных ответов.`

Методы:

`Statistic – конструктор по умолчанию;`

`void initialisation() – инициализация статистики из файла;`

`void rewriteInFile() const – перезапись файла статистики;`

`void setAmountAnswers(unsigned n) – установка количества ответов;`

`unsigned getAmountAnswers() – получение количества ответов;`

`void setAmountRightAnswers(unsigned n) – установка количества`

правильных ответов;

`unsigned getAmountRightAnswers() – получение количества`

правильных ответов;

`void setAmountDecks(unsigned n) – установка количества колод;`

`unsigned getAmountDecks() – получение количества колод;`

`void setAmountCards(unsigned n) – установка количества карточек;`

`unsigned getAmountCards() – получение количества карточек;`

`void setPercentage(float) – установка процента;`

`float getPercentage() const – получение процента.`

## 5. Класс SpacedRepetitionSystem

Класс системы интервального повторения. Содержит основные методы алгоритма повторения.

Методы:

`static void nextDateOfRepetition(Card*, bool) – расчёт следующей даты для повторения;`

```
static std::map <std::string, Deck*> createActiveDecks
(std::map <std::string, Deck*> ActiveDecks, const std::map
<std::string, Deck*>&Decks) – создание активных колод;
static std::list <Card*> createActiveCards ( std::list
<Card*> ActiveCards, const std::list <Card*> &Cards) – создание
списка активных карточек.
```

## 6. Класс DefaultCard

Класс обычной текстовой карточки. Унаследован от класса Card. Содержит текст для запоминания.

Поля:

std::string Text – текст карточки.

Методы:

DefaultCard – конструктор по умолчанию;

void setText(std::string text) – установка текста;

std::string getText() – получение текста;

void show() override – переопределение виртуальной функции.

## 7. Класс CardQ\_A

Класс карточки с вопросом и ответом. Унаследован от класса Card. Содержит вопрос и ответ на него.

Поля:

std::string Question – вопрос;

std::string Answer – ответ.

Методы:

CardQ\_A() – конструктор по умолчанию;

std::string getQuestion() – получение вопроса;

void setQuestion(std::string question) – установка вопроса;

std::string getAnswer() – получение ответа;

void setAnswer(std::string answer) – установка ответа;

void show() override – переопределение виртуальной функции.

## 8. Класс CardQ\_APicture

Класс карточки с вопросом с картинкой и ответом. Унаследован от класса CardQ\_A. Содержит путь к картинке.

Поля:

std::string AddressPicture – путь к картинке.

Методы:

CardQ\_APicture() – конструктор по умолчанию;

void setPicture(std::string address) – установка пути;

std::string getPicture() – получение пути;

`void show() override` – переопределение виртуальной функции.

## 9. Класс MyException

Класс исключений.

Поля:

`std::string ErrorMessage` – сообщение об ошибке.

Методы:

`explicit MyException(const std::string& errorMessage)` –  
конструктор;  
`void show() const` – метод отображения предупреждения.

## 10. Класс MenuWindow

Графический класс, отображающий главное меню приложения.

Поля:

`QPushButton* StartRepetition` – кнопка начала повторения;

`QPushButton *YourDeck` – кнопка перехода к колодам;

`QPushButton *Statistics` – кнопка статистики;

`QPushButton *Exit` – кнопка выхода.

Методы:

`void delay()` – метод задержки;

`explicit MenuWindow(QWidget* pwgt=nullptr)` – конструктор окна;

`void startRepetitionClicked()` – слот нажатия на кнопку  
`StartRepetition`;

`void yourDeckClicked()` – слот нажатия на кнопку `YourDeck`;

`void statisticsClicked()` – слот нажатия на кнопку `Statistics`;

`void exitClicked()` – слот нажатия на кнопку `Exit`.

## 11. Класс MainWindow

Графический класс, отображающий все колоды пользователя.

Поля:

`Session* session` – указатель на объект сессии;

`QPushButton* addDeck` – кнопка добавление колоды;

`QPushButton* deleteDeck` – кнопка удаления колоды;

`QPushButton* back` – кнопка возврата;

`QTableWidget* tblMain` – таблица колод;

`QHBoxLayout* hLayoutMain` – основной менеджер компоновки;

`int curRow` – текущий ряд элемента;

`int curColumn` – текущий столбец элемента.

### Методы:

`void delay()` – метод задержки;  
`explicit MainWindow(QWidget*pwgt=nullptr)` – конструктор окна;  
`void addDeckMain(const QString&)` – добавление колоды;  
`void renewTbl()` – обновление таблицы;  
`void takeFromSession()` – инициализация таблицы;  
`void returnBack()` – сигнал возврата;  
`void delDeckSlot()` – слот колоды;  
`void addDeckSlot()` – слот добавление колоды;  
`void deckClicked()` – слот выбора колоды;  
`void backClicked()` – слот возврата на предыдущую страницу.

## 12. Класс DeckWindow

Графический класс, отображающий все карточки определённой колоды.

### Поля:

`Deck*deck` – указатель на объект колоды;  
`QPushButton*addCard` – кнопка добавления карточки;  
`QPushButton*deleteCard` – кнопка удаления карточки;  
`QPushButton*back` – кнопка возврата;  
`QTableWidget*tblDeck` – таблица карточек;  
`QHBoxLayout*hLayoutDeck` – основной менеджер компоновки;  
`int curRow` – текущий ряд элемента;  
`int curColumn` – текущий столбец элемента.

### Методы:

`void delay()` – метод задержки;  
`explicit DeckWindow(QWidget*pwgt=nullptr, Deck*deck=nullptr)`  
– конструктор окна;  
`void addCardMain(const QString&)` – добавление карточки;  
`void returnBack()` – сигнал возврата;  
`void delCardSlot()` – слот удаления карточки;  
`void addCardSlot()` – слот добавления карточки;  
`void backClicked()` – слот возврата;  
`void editCardSlot()` – слот изменения карточки;  
`void initialisation()` – слот инициализации окна.

## 13. Класс DisplayCardWindow

Графический класс, отображающий содержимое карточки с возможностью редактирования.

### Поля:

`QPushButton*accept` – кнопка принятия изменений;  
`QPushButton*back` – кнопка возврата;

QLineEdit\*Name – поле ввода названия;  
QTextEdit\*Text – поле ввода текста;  
QTextEdit\*Question – поле ввода вопроса;  
QTextEdit\*Answer – поле ввода ответа;  
QLabel\*Picture – отображение картинки;  
Deck\*deck – указатель на объект колоды;  
Card\*card – указатель на объект карточки;  
std::string prevName – название карточки до изменения.

**Методы:**

void delay() – метод задержки;  
explicit DisplayCardWindow(QWidget\*wgt=nullptr, Card\*card=nullptr, Deck\*deck=nullptr) – конструктор окна;  
void returnBack() – сигнал возврата;  
void backClicked() – слот возврата;  
void saveName(const QString&) – слот сохранения названия;  
void saveText() – слот сохранения текста;  
void saveQuestion() – слот сохранения вопроса;  
void saveAnswer() – слот сохранения ответа;  
void savePicture() – слот сохранения картинки;  
void accepted() – слот принятия изменений.

## **14. Класс CreatingCardWindow**

Графический класс, отображающий окно с возможностью заполнения полей для создания карточки.

**Поля:**

Deck\*deck – указатель на объект колоды;  
QLineEdit\*Name – поле ввода имени;  
QTextEdit\*Text – поле ввода текста;  
QTextEdit\*Question – поле ввода вопроса;  
QTextEdit\*Answer – поле ввода ответа;  
QLabel\*Picture – отображение картинки;  
QToolButton\*ChooseFile – кнопка выбора картинки;  
QPushButton\*Accept – кнопка подтверждения создания;  
QPushButton\*Reject – кнопка отмены создания;  
QString AddressPicture – путь картинки;  
QString SavedText – сохраненный текст;  
QString SavedQuestion – сохраненный вопрос;  
QString SavedAnswer – сохраненный ответ.

**Методы:**

void delay() – метод задержки;  
void returnBack() – сигнал возврата;

```

explicit CreatingCardWindow (QWidget* wgt= nullptr,
Deck* deck= nullptr) – конструктор окна;
void accepted() – слот подтверждения;
void rejected() – слот отмены;
void textCard() – слот ввода текста;
void Q_ACard() – слот ввода вопроса/ответа;
void pictureChoose() – слот выбора картинки.

```

## 15. Класс StatisticsWindow

Графический класс, отображающий статистику пользователя.

Поля:

Statistics\*statistics – указатель на объект статистики.

Методы:

```

void delay() – метод задержки;
explicit StatisticsWindow (QWidget* wgt=nullptr) – конструктор
окна;
void returnBack() – сигнал возврата;
void backClicked() – слот возврата.

```

## 16. Класс RepetitionWindow

Графический класс, отображающий колоды, готовые для повторения.

Поля:

Session\*session – указатель на объект сессии;

QPushButton\*back – кнопка возврата;

QTableWidget\*tbl – таблица колод;

QHBoxLayout\*hLayout – основной менеджер компоновки;

QStackedWidget\*stack – стек виджетов;

int curRow=-1 – текущий ряд элемента;

int curColumn=0 – текущий столбец элемента.

Методы:

```

void delay() – метод задержки;
explicit RepetitionWindow (QWidget* wgt=nullptr) – конструктор
окна;
void takeFromSession() – инициализация окна;
void addDeck(const QString&) – добавление колоды;
void renewTbl() – обновление таблицы;
void intProcess() – процесс завершения повторения;
void returnBack() – сигнал возврата;
void deckClicked() – слот выбора колоды;
void backClicked() – слот возврата;
void changeWindow() – слот смены окна в стеке виджетов.

```

## 17. Класс RepetitionProcessWindow

Графический класс, отображающий определённую карточку из колоды в процессе повторения.

Поля:

Statistics\*statistics – указатель на объект статистики;

Deck\*deck – указатель на объект колоды;

Card\*ActiveCard – указатель на объект карточки;

std::string Type – тип карточки;

std::string SavedAnswer – сохранённый ответ.

Методы:

explicit RepetitionProcessWindow (QWidget \*wgt=nullptr, Card\*ActiveCard=nullptr, Deck\*deck=nullptr, Statistics\*statistics=nullptr) – конструктор окна;

void returnBack() – сигнал возврата;

void nextCard() – сигнал перехода к следующей карточке;

void checkClicked() – слот проверки;

void backClicked() – слот возврата;

void saveAnswer() – слот сохранения ответа.

## 18. Класс DeckButton

Графический класс, отображающий кликабельную ячейку (колоду или карточку).

Поля:

std::string Topic – тема.

Методы:

explicit DeckButton (QWidget \*wgt = nullptr) – конструктор кнопки;

std::string getTopic() – получение темы;

void setTopic (const QString&) – установка темы.

## 19. Класс InputNameWindow

Графический класс, отображающий поле для ввода темы колоды или название карточки.

Поля:

QLineEdit\* line – поле ввода;

QString str – название.

Методы:

explicit InputNameWindow (QWidget\*wgt=nullptr) – конструктор окна;

QString getTopic() – получение названия;



`void importText()` – слот сохранения текста.

## **20. Класс WarningWindow**

Графический класс, отображающий предупредительное окно при выходе из приложения.

Методы:

`explicit WarningWindow (QWidget*pwgt= nullptr)` – конструктор окна.

## **4 РАЗРАБОТКА П ПРОГРАММНЫХ МОДУЛЕЙ**

### **4.1 Разработка схем алгоритмов (два наиболее важных метода)**

В приложения Б и В представлены схемы методов `static void nextDateOfRepetition(Card*,bool)` и `static std::list<Card*> createActiveCards(std::list<Card*>, const std::list<Card*>&)` класса `SpacedRepetitionSystem` соответственно.

### **4.2 Разработка алгоритмов (описания алгоритмов по шагам, для двух методов)**

#### **4.2.1 Алгоритм проверки повторения `void checkClicked()` класса `RepetitionProcessWindow`**

Шаг 1. Начало;  
Шаг 2. Проверить тип карточки. Если карточка обычная – с текстом, переход к шагу 9;  
Шаг 3. Увеличить число ответов на 1;  
Шаг 4. Проверить правильность ответа. Если правильно, установить флаг `right` в `true`, иначе переход к шагу 8;  
Шаг 5. Вызов окна правильного ответа;  
Шаг 6. Увеличить число правильных ответов на 1, переход к шагу 10;  
Шаг 7. Установить флаг `right` в `false`;  
Шаг 8. Вызов окна неправильного ответа, переход к шагу 10;  
Шаг 9. Установить флаг `right` в `true`;  
Шаг 10. Рассчитать следующую дату повторения;  
Шаг 11. Перезаписать файл колоды;  
Шаг 12. Если число всех ответов не равно нулю, рассчитать процент правильных ответов;  
Шаг 13. Перезаписать файл статистики;  
Шаг 14. Конец.

#### **4.2.2 Алгоритм инициализации колоды `void initialisation()` класса `Deck`**

Шаг 1. Начало;  
Шаг 1. Открыть файл по указанному пути;  
Шаг 2. Считать строку файла. Если нарушена строка разделения “-----”, или достигнут конец файла, переход к шагу 8;  
Шаг 3. Создать массив строк;  
Шаг 4. Считать опорное слово;

Шаг 5. Считать информационную строку и записать в массив. Если она не пустая, удалить её первый элемент;

Шаг 6. Если массив не заполнен, переход к шагу 4;

Шаг 7. Вызвать метод создания карточки `createCard(std::string*)` класса `Deck`. Переход к шагу 2;

Шаг 8. Перезаписать файл колоды;

Шаг 9. Конец.

## 5 РЕЗУЛЬТАТЫ РАБОТЫ

Главное меню приложения с возможностью начать повторение, перейти к своим колодам, посмотреть статистику и выйти. Окно представлено на рисунке 5.1.

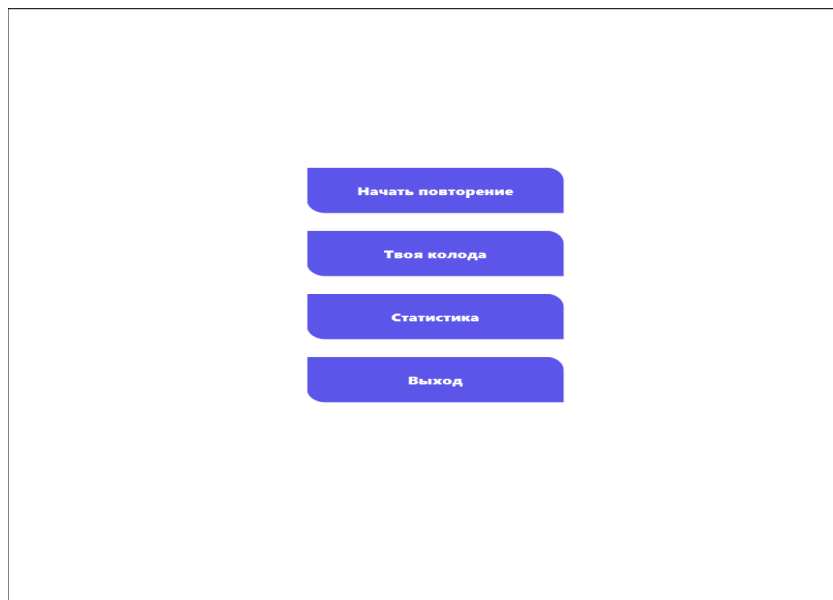


Рисунок 5.1 – Главное меню

Окно с размещением всех колод пользователя. Есть возможность вернуться, создать/удалить колоду и зайти в колоду. Окно представлено на рисунке 5.2.



Рисунок 5.2 – Окно с колодами

Окно с карточками определённой колоды, есть возможность вернуться, создать/удалить и просмотреть карточку. Окно представлено на рисунке 5.3.



Рисунок 5.3 – Окно с карточками

Окно с отображением содержимого карточки с возможностью редактирования. Окно представлено на рисунке 5.4.

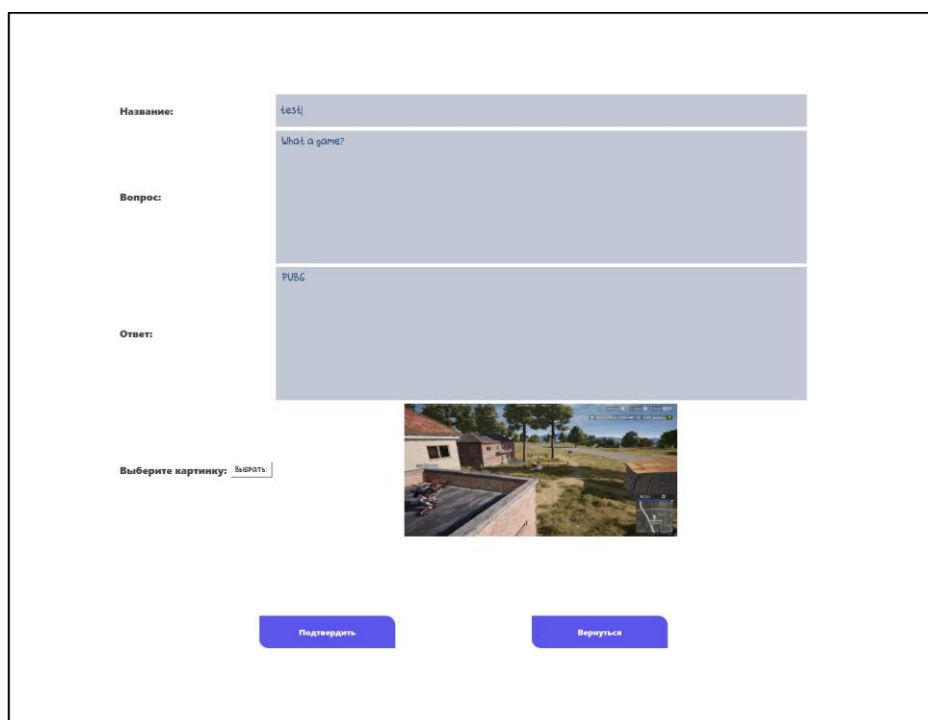


Рисунок 5.4 – Окно карточки

Окно создания карточки. Окно представлено на рисунке 5.5.



Рисунок 5.5 – Окно создания карточки

Окно с отображением процесса повторения карточки. Окно представлено на рисунке 5.6.

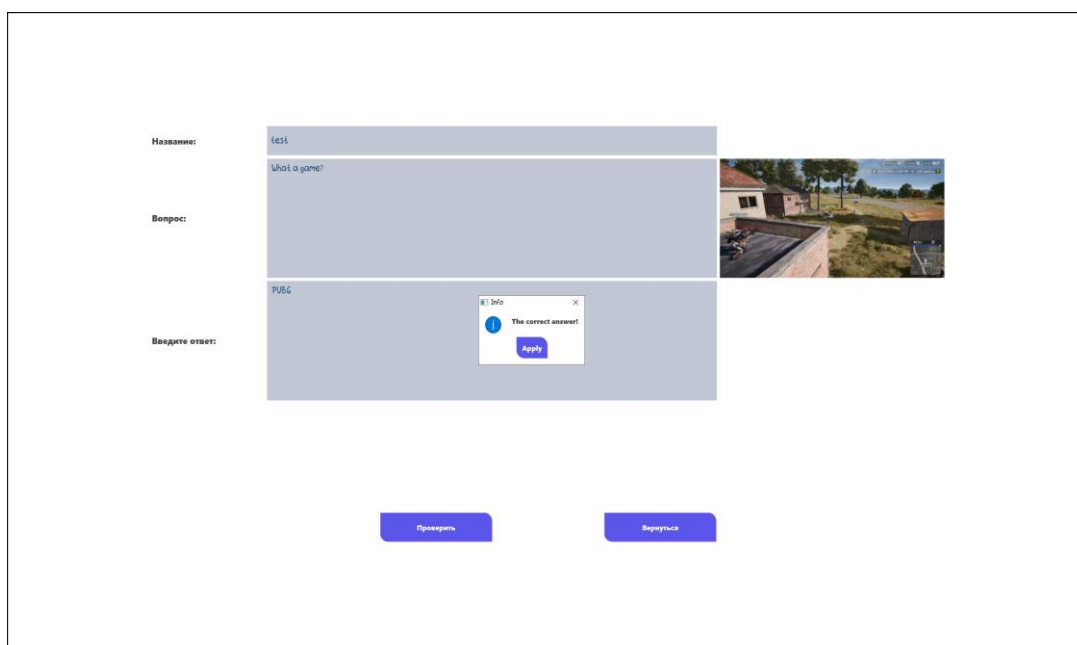


Рисунок 5.6 – Окно процесса повторения

Окно со статистикой повторения. Окно представлено на рисунке 5.7.

<b>Количество колод:</b>	<b>2</b>
<b>Количество карточек:</b>	<b>3</b>
<b>Количество ответов:</b>	<b>13</b>
<b>Правильные ответы:</b>	<b>10</b>
<b>Процент правильных ответов:</b>	<b>76.923103 %</b>
<div>Вернуться</div>	

Рисунок 5.7 – Окно статистики

## ЗАКЛЮЧЕНИЕ

В данном курсовом проекте было разработано приложение, представляющее собой обучающую программу с системой интервального повторения. Приложение создавалось по примеру наиболее популярных приложений такого формата, как “Anki” и “Quizlet”. В основном поставленные цели были реализованы.

Основными целями разработки обучающей программы было создание удобного и эффективного инструмента для изучения и запоминания информации, а также повышение эффективности образовательного процесса. Были определены требования к функциональности программы, такие как создание карточек с вопросами и ответами, определение оптимального интервала повторения, возможность работы с различными типами контента (текстом, изображениями и другими).

В ходе создания был в достаточной мере изучен функционал графической библиотеки Qt и выбраны наиболее подходящие ее средства. В процессе разработки использовались возможности языка C++, как объектно-ориентированного языка программирования.

Конечно же в дальнейшем хочется усовершенствовать приложение: добавить более качественное визуальное оформление, расширить функционал и придумать новые формы для запоминания.



## ЛИТЕРАТУРА

- [1] Anki [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://apps.ankiweb.net> –Дата доступа: 29.11.2023
- [2] Quizlet [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://quizlet.com/ru>.–Дата доступа: 29.11.2023
- [3] Шлее, М. Qt5.10. Профессиональное программирование на C++/ М.О. Шлее. – СПб, 2018. — 1072 с.
- [4] Луцик, Ю.А. Объектно–ориентированное программирование на языке C++: учеб. пособие /Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008.–266 с.
- [5] Страуструп, Б. Язык программирования C++ / Б. Страуструп специальное издание. Пер. с англ. – СПб.: BHV, 2008. – 1098 с.

**ПРИЛОЖЕНИЕ А**  
(обязательное)  
**Диаграмма классов**

## **ПРИЛОЖЕНИЕ Б**

(обязательное)

**Схема алгоритма метода `static void nextDateOfRepetition(Card*,bool)`  
класса `SpacedRepetitionSystem`**

## **ПРИЛОЖЕНИЕ В**

(обязательное)

**Схема алгоритма метода `static std::list<Card*>  
createActiveCards(std::list<Card*>, const std::list<Card*>&)` класса  
`SpacedRepetitionSystem`**

# ПРИЛОЖЕНИЕ Г

(обязательное)

## Исходный текст программы

### **main.cpp**

```
#include <iostream>
#include <QFile>
#include <QApplication>

#include "MenuWindow.h"
int main(int argc, char**argv)
{
    QApplication app(argc, argv);
    QFile file("Toolery.qss"); //открытие файла со стилем
    file.open(QFile::ReadOnly);
    QString qssStr=file.readAll();
    app->setStyleSheet(qssStr); //применение стиля

    MenuWindow* menuWindow=new MenuWindow; //создание окна
    menuWindow->setAttribute(Qt::WA_DeleteOnClose); //очистка
    памяти при закрытии окна
    menuWindow->showFullScreen(); //отображение окна в
    полноэкранном режиме

    return app.exec();
}
```

### **Session.h**

```
#pragma once
#include <map>
#include <QString>
#include <iostream>
#include <fstream>
#include <filesystem>

#include "Deck.h"
class Session{
private:
    Statistics*statistics;
    std::map<std::string, Deck*> Decks;
    std::map<std::string, Deck*> ActiveDecks;
public:
    Session();
    void createDeck(const std::string&,bool);
    void createFileDeck(const std::string&);
    void initialisation();
    std::map<std::string,Deck*> getDecks();
    std::map<std::string,Deck*> getActiveDecks();
    void setActiveDecks(std::map<std::string,Deck*>);
    void deleteDeck(const std::string&);
    static void deleteFileDeck(const std::string&);
    Statistics*getStatistics();
}
```

```

    ~Session();
};

```

### **Session.cpp**

```

#include "Session.h"

```

```

Session::Session()
{
    statistics=new Statistics;
}

```

```

Session::~~Session()
{
    delete statistics;
    for(const auto&item:Decks)
        delete item.second;
}

```

```

void Session::createDeck(const std::string& str,bool createFile)
{
    Deck *item=new Deck(statistics);
    item->setTopic(str);
    Decks[str]=item;
    if(createFile)
    {
        createFileDeck(str);
        return;
    }
    try
    {
        item->initialisation();
    }
    catch(const MyException& exception)
    {
        item->rewriteInFile();
        throw;
    }
}

```

```

void Session::createFileDeck(const std::string& str)
{
    statistics->setAmountDecks(statistics->getAmountDecks()+1);
    statistics->rewriteInFile();

    std::string filename = str + "Cards.txt";
    std::fstream file("D:/Coursework/Decks/" + filename,
std::ios::out); //открытие файла для записи
}

```

```

void Session::initialisation()
{

```

```

        statistics->initialisation();
        std::string path="D:/Coursework/Decks";
        for(const auto& entry:
std::filesystem::directory_iterator(path))    //проход по всем
файлам в директории
        {
            std::string str=entry.path().filename().string();
            std::string::size_type pos1=str.find("Cards.txt");
            if(pos1!=std::string::npos)    //если найдено
            {
                str.erase(pos1,9);
                std::string::size_type pos2=str.find(".txt");
                if(pos2==std::string::npos && !str.empty())
                    createDeck(str, false);
            }
        }
    }

std::map<std::string ,Deck*> Session::getDecks()
{
    return Decks;
}

void Session::setActiveDecks(std::map<std::string, Deck
*>activeDecks)
{
    ActiveDecks=activeDecks;
}

std::map<std::string,Deck*> Session::getActiveDecks()
{
    return ActiveDecks;
}

void Session::deleteDeck(const std::string& str)
{
    for(auto pos=Decks.begin();pos!=Decks.end();)
    {
        if(str==pos->first)
        {
            for(const auto& item:pos->second->getCards())
            {
                pos->second->deleteCard(item->getName());
            }
            pos=Decks.erase(pos);
            statistics->setAmountDecks(statistics-
>getAmountDecks()-1);
            statistics->rewriteInFile();
            deleteFileDeck(str);
        }
        else
            ++pos;
    }
}

```

```

    }
}

void Session::deleteFileDeck(const std::string& str)
{
    std::string path="D:/Coursework/Decks";
    for(const auto& entry:
std::filesystem::directory_iterator(path))    //проход по файлам
директории
    {
        std::string filename=str+"Cards.txt";
        if(filename==entry.path().filename().string())
//получение имени файла
        {
            std::string full=path+"/"+filename;
            remove(full.c_str());    //удаление
        }
    }
}

Statistics* Session::getStatistics()
{
    return statistics;
}

```

### **Deck.h**

```

#pragma once
#include <list>
#include <iostream>
#include <QDate>
#include <QTime>
#include <regex>
#include <fstream>

#include "DefaultCard.h"
#include "CardQ_APicture.h"
#include "Statistics.h"
#include "MyException.h"

class Deck{
private:
    Statistics*statistics;
    std::string Topic;
    std::list<Card*> Cards;
    std::list<Card*> ActiveCards;
public:
    explicit Deck(Statistics*);
    ~Deck();
    void createCard(std::string*,bool);
    void createFileCard(std::string*);
    void initialisation();
    void deleteCard(const std::string&);

```



```

    void setTopic(std::string);
    std::list<Card*> getCards();
    void setActiveCards(std::list<Card*>);
    std::list<Card*> getActiveCards();
    std::string static replaceAll(std::string&, const
std::string&, const std::string&);
    std::string static
readAnyString(std::ifstream&, char*, std::string );
    void rewriteInFile() const;
};

```

### **Deck.cpp**

```

#include "Deck.h"

Deck::Deck(Statistics*stats)
{
    statistics=stats;
    Topic="";
}

Deck::~~Deck()
{
    qDeleteAll(Cards); //удаления всех карточек
}

void Deck::setTopic(std::string topic)
{
    Topic=topic;
}

void Deck::createCard(std::string *inf, bool addToFile)
{
    if(inf[2].empty() && inf[3].empty() && inf[4].empty() )
    {
        Card*card=new DefaultCard;
        auto*defCard=dynamic_cast<DefaultCard*>(card);
        card->setType("DefaultCard");
        defCard->setName(inf[0]);
        inf[1]= replaceAll(inf[1], "\n", "[enter]"); //замена
всех '\n' на [enter]
        defCard->setText(inf[1]);
        defCard->setNextDateOfRepetition(inf[5]);
        defCard->setNextTimeOfRepetition(inf[6]);
        defCard->setLevelOfMemorization(std::stoi(inf[7]));
        defCard->setActivity("True"==inf[8]);
        Cards.push_back(card);
    }

    else if(inf[1].empty() && inf[4].empty()){
        Card*card=new CardQ_A;
        auto*cardQ_A=dynamic_cast<CardQ_A*>(card);
        card->setType("CardQ_A");
        cardQ_A->setName(inf[0]);
    }
}

```

```

        inf[2]= replaceAll(inf[2], "\n", "[enter]");
        cardQ_A->setQuestion(inf[2]);
        inf[3]= replaceAll(inf[3], "\n", "[enter]");
        cardQ_A->setAnswer(inf[3]);
        cardQ_A->setNextDateOfRepetition(inf[5]);
        cardQ_A->setNextTimeOfRepetition(inf[6]);
        cardQ_A->setLevelOfMemorization(std::stoi(inf[7]));
        cardQ_A->setActivity("True"==inf[8]);
        Cards.push_back(card);
    }

    else if( inf[1].empty())
    {
        Card*card=new CardQ_APicture;
        auto*cardQ_APicture=dynamic_cast<CardQ_APicture*>(card);
        card->setType("CardQ_APicture");
        cardQ_APicture->setName(inf[0]);
        inf[2]= replaceAll(inf[2], "\n", "[enter]");
        cardQ_APicture->setQuestion(inf[2]);
        inf[3]= replaceAll(inf[3], "\n", "[enter]");
        cardQ_APicture->setAnswer(inf[3]);
        cardQ_APicture->setPicture(inf[4]);
        cardQ_APicture->setNextDateOfRepetition(inf[5]);
        cardQ_APicture->setNextTimeOfRepetition(inf[6]);
        cardQ_APicture->setLevelOfMemorization(std::stoi(inf[7]));
        cardQ_APicture->setActivity("True"==inf[8]);
        Cards.push_back(card);
    }

    if (addToFile) {
        statistics->setAmountCards(statistics->getAmountCards() + 1);
        statistics->rewriteInFile();
        createFileCard(inf);
    }
}

void Deck::createFileCard( std::string *inf)
{
    std::string filename= Topic+"Cards.txt";
    std::fstream file("D:/Coursework/Decks/"+filename, std::ios
::app); //открытие потока для дозаписи
    if(file.fail())
        throw MyException("Unable to open a deck file:
"+filename);

    file<<"-----"<<std::endl;
    file<<"Name: "+inf[0]<<std::endl;
    file<<"Text: "+inf[1]<<std::endl;
    file<<"Question: "+inf[2]<<std::endl;
    file<<"Answer: "+inf[3]<<std::endl;
    file<<"AddressPicture: "+inf[4]<<std::endl;

```

```

        file<<"NextDateOfRepetition: "+inf[5]<<std::endl;
        file<<"NextTimeOfRepetition: "+inf[6]<<std::endl;
        file<<"LevelOfMemorization: "+inf[7]<<std::endl;
        file<<"Activity: "+inf[8]<<std::endl;
    }

void Deck::deleteCard(const std::string& name)
{
    for(auto pos=Cards.begin();pos!=Cards.end();pos++) //проход по
    листу
    {
        if((*pos)->getName()==name)
        {
            statistics->setAmountCards(statistics-
            >getAmountCards()-1);
            statistics->rewriteInFile();
            pos=Cards.erase(pos); //удаление карточки
            rewriteInFile(); //перезапись файла колоды
        }
        else
            ++pos;
    }
}

void Deck::initialisation()
{
    std::string filename = Topic + "Cards.txt";
    std::ifstream file("D:/Coursework/Decks/"+filename);
    if(file.fail())
        throw MyException("Unable to open a deck file!
        "+filename);
    while(true)
    {
        char buf[255];
        file.getline(buf, 255);
        if(file.eof())
            break;
        if((std::string)buf!="-----" )
            throw MyException("Unable to read a file:
            "+filename);
        std::string inf[9];
        for(auto & i : inf)
        {
            std::string str;
            file>>buf;
            str=static_cast<std::string>(buf);
            if(str.empty())
                throw MyException("Unable to read a file:
                "+filename);
            i=readAnyString(file,buf,i);
            if (!i.empty())
                i.erase(0, 1);
        }
    }
}

```

```

        }
        createCard(inf, false);
    }
    rewriteInFile();
}

std::list<Card*> Deck::getCards()
{
    return Cards;
}

void Deck::setActiveCards(std::list<Card *>activeCards)
{
    ActiveCards=activeCards;
}

std::list<Card*> Deck::getActiveCards()
{
    return ActiveCards;
}

std::string Deck::replaceAll(std::string &str, const std::string
&src, const std::string &dst)
{
    std::regex rx(src.c_str());
    return std::regex_replace(str, rx, dst);
}

std::string
Deck::readAnyString(std::ifstream&file, char*buf, std::string res)
{
    bool ff, fe;
    res.clear();
    while(true) {
        file.getline(buf, 255);
        ff=file.fail(); //флаг ошибка
        fe=file.eof(); //флаг конца файла
        res += buf;
        if(fe)
            break;
        if(ff) {
            file.clear(); //очистка флагов
            continue;
        }
        else
            break;
    }
    return res;
}

void Deck::rewriteInFile() const
{

```

```

std::string path="D:/Coursework/Decks/"+Topic+"Cards.txt";
std::ofstream file(path);

for(const auto& item:Cards)
{
    file<<"-----"<<std::endl;
    file<<"Name: "+item->getName()<<std::endl;
    std::string str;
    if(item->getType()=="DefaultCard")
    {
        str=dynamic_cast<DefaultCard*>(item)->getText();
    }
    file<<"Text: "+str<<std::endl;

    if(item->getType()=="CardQ_A")
    {
        str=dynamic_cast<CardQ_A*>(item)->getQuestion();
        file<<"Question: "+str<<std::endl;
        str=dynamic_cast<CardQ_A*>(item)->getAnswer();
        file<<"Answer: "+str<<std::endl;
        file<<"AddressPicture: "<<std::endl;
    }
    else if(item->getType()=="CardQ_APicture")
    {
        str=dynamic_cast<CardQ_APicture*>(item)-
>getQuestion();
        file<<"Question: "+str<<std::endl;
        str=dynamic_cast<CardQ_APicture*>(item)-
>getAnswer();
        file<<"Answer: "+str<<std::endl;
        str=dynamic_cast<CardQ_APicture*>(item)-
>getPicture();
        file<<"AddressPicture: "+str<<std::endl;
    }
    else
    {
        file<<"Question: "<<std::endl;
        file<<"Answer: "<<std::endl;
        file<<"AddressPicture: "<<std::endl;
    }
    file<<"NextDateOfRepetition: "+item-
>getNextDateOfRepetition()<<std::endl;
    file<<"NextTimeOfRepetition: "+item-
>getNextTimeOfRepetition()<<std::endl;
    file<<"LevelOfMemorization: "+std::to_string(item-
>getLevelOfMemorization())<<std::endl;
    if(item->getActivity())
        file<<"Activity: True"<<std::endl;
    else
        file<<"Activity: False"<<std::endl;
    }
}

```

### **Card.h**

```
#pragma once
#include <iostream>

class Card{
private:
    std::string type;
protected:
    std::string Name;
    unsigned LevelOfMemorization;
    std::string NextTimeOfRepetition;
    std::string NextDateOfRepetition;
    bool Activity;
public:
    Card(): LevelOfMemorization(0), Activity(false) {};
    virtual ~Card();
    virtual void show()=0;
    std::string getName();
    void setName(std::string);
    unsigned getLevelOfMemorization() const;
    void setLevelOfMemorization(unsigned );
    std::string getNextTimeOfRepetition();
    void setNextTimeOfRepetition(std::string);
    std::string getNextDateOfRepetition();
    void setNextDateOfRepetition(std::string);
    bool getActivity() const;
    void setActivity(bool );
    void setType(std::string);
    std::string getType();
};
```

### **Card.cpp**

```
#include "Card.h"

Card::~Card()= default;

std::string Card::getName()
{
    return Name;
}

void Card::setName(std::string name)
{
    Name=name;
}

unsigned Card::getLevelOfMemorization() const
{
    return LevelOfMemorization;
}
```

```

void Card::setLevelOfMemorization(unsigned int level)
{
    LevelOfMemorization=level;
}

std::string Card::getNextTimeOfRepetition()
{
    return NextTimeOfRepetition;
}

void Card::setNextTimeOfRepetition(std::string time)
{
    NextTimeOfRepetition=time;
}

std::string Card::getNextDateOfRepetition()
{
    return NextDateOfRepetition;
}

void Card::setNextDateOfRepetition(std::string date)
{
    NextDateOfRepetition=date;
}

bool Card::getActivity() const
{
    return Activity;
}

void Card::setActivity(bool act)
{
    Activity = act;
}

void Card::setType(std::string str)
{
    type=str;
}

std::string Card::getType()
{
    return type;
}

```

#### **CardQ\_A.h**

```

#pragma once
#include "Card.h"

```

```

class CardQ_A: public Card{
    std::string Question;
    std::string Answer;
}

```

```

public:
    CardQ_A():Card(){};
    std::string getQuestion();
    void setQuestion(std::string question);
    std::string getAnswer();
    void setAnswer(std::string answer);
    void show() override;
};

```

#### **CardQ\_A.cpp**

```

#include "CardQ_A.h"

std::string CardQ_A::getQuestion()
{
    return Question;
}

void CardQ_A::setQuestion(std::string question)
{
    Question=question;
}

std::string CardQ_A::getAnswer()
{
    return Answer;
}

void CardQ_A::setAnswer(std::string answer)
{
    Answer=answer;
}

void CardQ_A::show() {};

```

#### **CardQ\_APicture.h**

```

#pragma once
#include "CardQ_A.h"

class CardQ_APicture: public CardQ_A{
    std::string AddressPicture;
public:
    CardQ_APicture():CardQ_A(){};
    void setPicture(std::string address);
    std::string getPicture();
    void show() override;
};

```

#### **CardQ\_APicture.cpp**

```

#include "CardQ_APicture.h"

std::string CardQ_APicture::getPicture()

```



```

{
    return AddressPicture;
}

void CardQ_APicture::setPicture(std::string address)
{
    AddressPicture=address;
}

void CardQ_APicture::show() {};

Statistics.h
#pragma once
#include <fstream>
#include <sstream>

#include "MyException.h"

class Statistics{
private:
    unsigned AmountDecks;
    unsigned AmountCards;
    unsigned AmountAnswers;
    unsigned AmountRightAnswers;
    float Percentage;
public:
    Statistics(): AmountDecks(0), AmountAnswers(0),
    AmountRightAnswers(0), AmountCards(0),Percentage(0.0){};
    void initialisation();
    void addStatistics(std::string*);
    void rewriteInFile() const;
    void setAmountDecks(unsigned );
    unsigned getAmountDecks() const;
    void setAmountCards(unsigned);
    unsigned getAmountCards() const;
    void setAmountAnswers(unsigned);
    unsigned getAmountAnswers() const;
    void setAmountRightAnswers(unsigned);
    unsigned getAmountRightAnswers() const;
    void setPercentage(float);
    float getPercentage() const;
};

Statistics.cpp
#include "Statistics.h"

void Statistics::initialisation()
{
    std::string path="D:/Coursework/Statistics.txt";
    std::ifstream file(path);
    if(file.fail())

```

```

    {
        throw MyException("Unable to initialise statistics!");
    }
    std::string inf[5];
    char buf[255];
    for(auto & i : inf)
    {
        file>>buf;
        file.getline(buf, 254);
        i=buf;
        if(i.empty())
            throw MyException("Unable to initialise
statistics!");
        i.erase(0, 1);
    }
    addStatistics(inf);
}

void Statistics::addStatistics(std::string *inf)
{
    //установка данных
    setAmountDecks(std::stoi(inf[0]));
    setAmountCards(std::stoi(inf[1]));
    setAmountAnswers(std::stoi(inf[2]));
    setAmountRightAnswers(std::stoi(inf[3]));
    setPercentage(std::stof(inf[4]));
}

void Statistics::rewriteInFile() const
{
    std::string path="D:/Coursework/Statistics.txt";

    std::ofstream newFile(path);

    std::ostringstream oss; //поток ввода
    oss<<getAmountDecks(); //внесение в поток
    newFile<<"AmountDecks: "+oss.str()<<std::endl;
    oss.str(""); //очистка поток
    oss<<getAmountCards();
    newFile<<"AmountCards: "+oss.str()<<std::endl;
    oss.str("");
    oss<<getAmountAnswers();
    newFile<<"AmountAnswers: "+oss.str()<<std::endl;
    oss.str("");
    oss<<getAmountRightAnswers();
    newFile<<"AmountRightAnswers: "+oss.str()<<std::endl;
    oss.str("");
    oss<<getPercentage();
}

```

```

        newFile<<"Percentage: "+oss.str()<<std::endl;

    }

    void Statistics::setAmountDecks(unsigned int amount)
    {
        AmountDecks=amount;
    }

    unsigned Statistics::getAmountDecks() const
    {
        return AmountDecks;
    }

    void Statistics::setAmountCards(unsigned int amount)
    {
        AmountCards=amount;
    }

    unsigned Statistics::getAmountCards() const
    {
        return AmountCards;
    }

    void Statistics::setAmountAnswers(unsigned int amount)
    {
        AmountAnswers=amount;
    }

    unsigned Statistics::getAmountAnswers() const
    {
        return AmountAnswers;
    }

    void Statistics::setAmountRightAnswers(unsigned int amount)
    {
        AmountRightAnswers=amount;
    }

    unsigned Statistics::getAmountRightAnswers() const
    {
        return AmountRightAnswers;
    }

    void Statistics::setPercentage(float percentage)
    {
        Percentage=percentage;
    }

```

```
float Statistics::getPercentage() const
{
    return Percentage;
}
```

### **SpacedRepetitionSystem.h**

```
#pragma once
#include <QDate>
#include <fstream>
#include <sstream>
#include <cstdio>
#include <windows.h>
#include <cmath>

#include "Deck.h"

class SpacedRepetitionSystem {
public:
    static void nextDateOfRepetition(Card*, bool);
    static std::map<std::string, Deck*>
createActiveDecks(std::map<std::string, Deck*> ActiveDecks, const
std::map<std::string, Deck*>&Decks);
    static std::list<Card*>
createActiveCards(std::list<Card*>ActiveCards, const
std::list<Card*>&Cards);
};
```

### **SpacedRepetitionSystem.cpp**

```
#include "SpacedRepetitionSystem.h"

void SpacedRepetitionSystem::nextDateOfRepetition(Card
*card, bool right)
{
    if (right)
    {
        unsigned interval;
        card->setActivity(false);
        if (card->getLevelOfMemorization()==0)
            interval=1;
        else {
            unsigned fromFirstRepetitionThis = 2 * (pow(2,
card->getLevelOfMemorization()) - 1) + 1;
            unsigned fromFirstRepetitionPrev= 2 *
(pow(2, card->getLevelOfMemorization()-1)-1)+1;
            interval=fromFirstRepetitionThis-
fromFirstRepetitionPrev;
        }

        QDate Qdate=QDate::currentDate().addDays(interval);
```

```

//добавление интервала к текущей дате
    card-
>setNextDateOfRepetition(Qdate.toString("dd.MM.yyyy").toStdString()); //установка даты

    card->setLevelOfMemorization(card-
>getLevelOfMemorization()+1); //установка уровня запоминания
    }
    else
    {
        card->setActivity(true);

        QDate Qdate=QDate::currentDate();
        card-
>setNextDateOfRepetition(Qdate.toString("dd.MM.yyyy").toStdString());

        card->setLevelOfMemorization(0);
    }
}

std::map<std::string, Deck*>
SpacedRepetitionSystem::createActiveDecks(std::map<std::string, Deck*> ActiveDecks, const std::map<std::string, Deck*>&Decks)
{
    ActiveDecks.clear(); //очистка листа
    for(const auto &item:Decks)
    {
        item.second-
>setActiveCards(SpacedRepetitionSystem::createActiveCards(item.second->getActiveCards(), item.second->getCards()));
        if(!item.second->getActiveCards().empty())
        {
            ActiveDecks[item.first]=item.second;
        }
        item.second->rewriteInFile();
    }
    return ActiveDecks;
}

std::list<Card*>SpacedRepetitionSystem::createActiveCards(std::list<Card*>ActiveCards, const std::list<Card*>&Cards)
{
    ActiveCards.clear();
    QDate QCurrentdate=QDate::currentDate();
    std::string
date=(QCurrentdate.toString("dd.MM.yyyy")).toStdString();

```

```

        for(const auto&item:Cards)
        {
            QString StringDate=StringDate.fromString(item-
>getNextDateOfRepetition());

            QDate Qdate=QDate::fromString(StringDate,"dd.MM.yyyy");
            if(QCurrentdate>=Qdate )
            {
                item->setActivity(true);
                item->setNextDateOfRepetition(date);

                ActiveCards.push_back(item);
            }
        }
        return ActiveCards;
    }
}

```

### **MyException.h**

```

#pragma once
#include <string>
#include <QMessageBox>

class MyException{
private:
    std::string ErrMessage;
public:
    explicit MyException(const std::string&
ErrMessage):ErrMessage(ErrMessage){};
    void show() const
    {
        QString string=string.fromString(ErrMessage);
        while (QMessageBox::warning(nullptr, "Error ", string,
QMessageBox::Apply) !=QMessageBox::Apply);
    }
};

```

### **MenuWindow.h**

```

#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QPushButton>
#include <QDialog>
#include <windows.h>
#include <winuser.h>

#include "WarningWindow.h"
#include "MainWindow.h"
#include "RepetitionWindow.h"

```

```

#include "StatisticsWindow.h"

class MenuWindow:public QWidget{
    Q_OBJECT
private:
    QPushButton *StartRepetition;
    QPushButton *YourDeck;
    QPushButton *Statistics;
    QPushButton *Exit;
    void delay();
public:
    explicit MenuWindow(QWidget*pwgt=nullptr);
public slots:
    void startRepetitionClicked();
    void yourDeckClicked();
    void statisticsClicked();
    void exitClicked();
};

MenuWindow.cpp
#include "MenuWindow.h"

MenuWindow::MenuWindow(QWidget*pwgt): QWidget(pwgt)
{
    //получение размеров экрана в пикселях
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    StartRepetition=new QPushButton("Начать повторение");
    //создание кнопки
    StartRepetition-
>setSizePolicy(QSizePolicy::Preferred,QSizePolicy::Preferred);
    //установка политики расширения
    StartRepetition->setMaximumSize(0.104*width,0.046*height);
    //установка максимального размера

    YourDeck=new QPushButton("Твоя колода");
    YourDeck-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    YourDeck->setMaximumSize(0.104*width,0.046*height);

    Statistics=new QPushButton("Статистика");
    Statistics-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Statistics->setMaximumSize(0.104*width,0.046*height);

    Exit=new QPushButton("Выход");
    Exit-

```

```

>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Exit->setMaximumSize(0.104*width,0.046*height);

    auto* vLayout=new QVBoxLayout; //создание менеджера
компоновки
    vLayout->setAlignment(Qt::AlignCenter); //выравнивание
элементов layout по центру
    vLayout->setSpacing(0.01*width);
    vLayout->addWidget(StartRepetition);
    vLayout->addWidget(YourDeck);
    vLayout->addWidget(Statistics);
    vLayout->addWidget(Exit);
    setLayout(vLayout); //установка layout как главного

connect(StartRepetition,SIGNAL(clicked()),SLOT(startRepetitionCl
icked())); //связывание нажатия кнопки с переходом на следующее
окно
    connect(YourDeck,SIGNAL(clicked()),SLOT(yourDeckClicked()));

connect(Statistics,SIGNAL(clicked()),SLOT(statisticsClicked()));
    connect(Exit, SIGNAL(clicked()),SLOT(exitClicked()));
}

void MenuWindow::yourDeckClicked()
{
    try
    {
        auto *mainWindow = new MainWindow;
        mainWindow->setAttribute(Qt::WA_DeleteOnClose);
//удаление окна при закрытии
        mainWindow->showFullScreen();
        connect(mainWindow, &MainWindow::returnBack, this,
&MenuWindow::show);
        delay(); //создание задержки
        hide(); //спрятать окно
    }
    catch(const MyException& exception)
    {
        exception.show();
    }
    catch(...)
    {
        while (QMessageBox::warning(nullptr, "Error", "Unknown
error!", QMessageBox::Apply) !=
            QMessageBox::Apply);
    }
}

```



```

}

void MenuWindow::exitClicked()
{
    auto*warning=new WarningWindow;
    warning->setAttribute(Qt::WA_DeleteOnClose);
    if(warning->exec()==QDialog::Accepted)    //отображение окна
приоритетно
    {
        close();
    }
}

void MenuWindow::startRepetitionClicked()
{
    try {
        auto *repetitionWindow = new RepetitionWindow;
        repetitionWindow->setAttribute(Qt::WA_DeleteOnClose);
        repetitionWindow->showFullScreen();
        connect(repetitionWindow, &RepetitionWindow::returnBack,
this, &MenuWindow::show);
        delay();
        hide();
    }
    catch(const MyException& exception)
    {
        exception.show();
    }
    catch(...)
    {
        while (QMessageBox::warning(nullptr, "Error", "Unknown
error!", QMessageBox::Apply) !=
            QMessageBox::Apply);
    }
}

void MenuWindow::statisticsClicked()
{
    try
    {
        auto *statisticsWindow = new StatisticsWindow;
        statisticsWindow->setAttribute(Qt::WA_DeleteOnClose);
        statisticsWindow->showFullScreen();
        connect(statisticsWindow, &StatisticsWindow::returnBack,
this, &MenuWindow::show);
        delay();
        hide();
    }
}

```

```

        catch(const MyException& exception)
        {
            exception.show();
        }
        catch(...)
        {
            while (QMessageBox::warning(nullptr, "Error", "Unknown
error!", QMessageBox::Apply) !=
                QMessageBox::Apply);
        }
    }

void MenuWindow::delay()
{
    QEventLoop loop; //цикл
    QTimer timer; //таймер
    timer.setInterval(100); //0.1sec
    connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
    timer.start(); //запуск таймера
    loop.exec(); //начало цикла повторений
}

```

### **MainWindow.h**

```

#pragma once
#include <QWidget>
#include <QPushButton>
#include <QTableWidget>
#include <map>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QHeaderView>
#include <QLabel>
#include <QMessageBox>

#include "Session.h"
#include "DeckButton.h"
#include "DeckWindow.h"

class MainWindow:public QWidget{
    Q_OBJECT
private:
    Session*session;
    QPushButton*addDeck;
    QPushButton*deleteDeck;
    QPushButton*back;
    QTableWidget*tblMain;

```

```

        QHBoxLayout*hLayoutMain;
        int curRow=-1;
        int curColumn=0;

        void delay();
public:
        explicit MainWindow(QWidget*pwgt=nullptr);
        ~MainWindow(){delete session;}
        void addDeckMain(const QString&);
        void renewTbl();
        void takeFromSession();
signals:
        void returnBack();
public slots:
        void delDeckSlot();
        void addDeckSlot();
        void deckClicked();
        void backClicked();
};

```

### **MainWindow.cpp**

```

#include "MainWindow.h"

MainWindow::MainWindow(QWidget *pwgt): QWidget(pwgt)
{
    //получение рвзмеров экрана в пикселях
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    addDeck=new QPushButton("Добавить колоду");
    addDeck-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    addDeck->setMaximumSize(0.104*width,0.046*height);

    deleteDeck=new QPushButton("Удалить колоду");
    deleteDeck-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    deleteDeck->setMaximumSize(0.104*width,0.046*height);

    back=new QPushButton("Вернуться");
    back-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    back->setMaximumSize(0.104*width,0.046*height);

    tblMain=new QTableWidgetItem(0,9);
    tblMain->setMaximumSize(0.753*width,0.815*height);
    tblMain->horizontalHeader()->setVisible(false); //спрятать

```

```

горизонтальный заголовок
    tblMain->verticalHeader()->setVisible(false); // спрятать
вертикальный заголовок
    tblMain->setShowGrid(false); //спрятать сетку
    tblMain->setSelectionMode(QAbstractItemView::NoSelection);
//отключить выбор
    tblMain->setEditTriggers(QAbstractItemView::NoEditTriggers);
//отключить выделение
    tblMain->setFocusPolicy(Qt::NoFocus); //отключить фокус
ячейки
    tblMain->setStyleSheet("QTableWidget::item{padding: 10px}");

    auto*vLayoutMain=new QVBoxLayout;
    vLayoutMain->setAlignment(Qt::AlignTop);
    vLayoutMain->addSpacing(0.047*height);
    vLayoutMain->setSpacing(0.01*width);
    vLayoutMain->addWidget(back);
    vLayoutMain->addWidget(addDeck);
    vLayoutMain->addWidget(deleteDeck);

    hLayoutMain=new QHBoxLayout;
    hLayoutMain->addLayout(vLayoutMain);
    hLayoutMain->addWidget(tblMain);
    setLayout(hLayoutMain);

    try {
        session = new Session;
        session->initialisation(); //инициализация сессии
        takeFromSession();
    }
    catch( const MyException& exception)
    {

        throw;
    }
    catch(...)
    {
        throw;
    }

    //связывание нажатий кнопок с переходами на другие окна
    connect(addDeck, SIGNAL(clicked()),SLOT(addDeckSlot()));
    connect(deleteDeck,SIGNAL(clicked()),SLOT(delDeckSlot()));
    connect(back,SIGNAL(clicked()),SLOT(backClicked()));
}

```

```

void MainWindow::addDeckMain(const QString& string)

```

```

{
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();
    if(curColumn==0)
    {
        tblMain->setRowCount(tblMain->rowCount() + 1);
        curRow++;
        tblMain->setRowHeight(tblMain->rowCount() - 1,
0.139*height);
    }
    if (curColumn == tblMain->columnCount())
    {
        tblMain->setRowCount(tblMain->rowCount() + 1);
        curRow++;
        tblMain->setRowHeight(tblMain->rowCount() - 1,
0.139*height);
        curColumn = 0;
    }
    tblMain->setColumnWidth(curColumn, 0.083*width);
    auto*wgt=new QWidget; //создание кнопки
    auto *item = new DeckButton(); //создание кнопки
    item->setTopic(string);
    item-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    // связывание нажатие на кнопку с открытием колоды
    connect(item, SIGNAL(clicked()),SLOT(deckClicked()));

    auto*label=new QLabel(string);

    label->setAlignment(Qt::AlignHCenter | Qt::AlignBottom);
    auto*vLayout=new QVBoxLayout;
    label->setBuddy(item);
    vLayout->addWidget(item);
    vLayout->addWidget(label);
    wgt->setLayout(vLayout);

    tblMain->setCellWidget(curRow, curColumn, wgt);
    curColumn++;
}

void MainWindow::addDeckSlot()
{
    QString string;
    auto *topic = new InputNameWindow(nullptr);
    if (topic->exec() == QDialog::Accepted)
    {

```

```

        string = topic->getTopic();
    }
    topic->close();
    delete topic;
    if(!string.isEmpty()) {
        session->createDeck(string.toStdString(),true);
        renewTbl();
        takeFromSession();
    }
}

void MainWindow::renewTbl()
{
    for(int i=0;i<tblMain->rowCount();i++)
        for(int j=0;j<tblMain->columnCount();j++)
        {
            if(i==curColumn && j==curColumn)
            {
                i=tblMain->rowCount();
                break;
            }
            tblMain->removeCellWidget(i,j); //удаление ячейки
        }

    curRow=-1;
    curColumn=0;
    tblMain->setRowCount(0);
    tblMain->setColumnCount(9);
}

void MainWindow::takeFromSession()
{
    std::map<std::string ,Deck*> Copy=session->getDecks();
    for(const auto &item: Copy)
    {
        std::string str=item.first;
        QString string;
        string=string.fromStdString(str); //приведение к типу
        QString
        addDeckMain(string); //добавление ячейку
    }
}

```

```

void MainWindow::delDeckSlot()
{
    QString str;
    auto*topic=new InputNameWindow;
    if(topic->exec()==QDialog::Accepted)    //вызов окна
приоритетно
    {
        str=topic->getTopic();
    }
    topic->close();
    delete topic;
    if(!str.isEmpty())    //если строка не пустая
    {
        session->deleteDeck(str.toStdString());
        renewTbl();
        takeFromSession();
    }
}

void MainWindow::deckClicked()
{
    auto*button= qobject_cast<DeckButton*>(sender());
    std::string str=button->getTopic();

    std::map<std::string,Deck*>Decks=session->getDecks();
    auto*deckWindow=new DeckWindow(nullptr,Decks[str]);
    deckWindow->setAttribute(Qt::WA_DeleteOnClose);
    deckWindow->showFullScreen();
    connect(deckWindow,
&DeckWindow::returnBack,this,&MainWindow::show);
    delay();
    hide();
}

void MainWindow::backClicked()
{
    emit returnBack(); // вызов сигнала
    delay();    //задержка
    close();
}

void MainWindow::delay()
{
    QEventLoop loop;
    QTimer timer;
    timer.setInterval(100); //0.1sec

```

```

        connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
        timer.start();
        loop.exec();
    }

```

### **DisplayCardWindow.h**

```

#pragma once
#include <QWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QFormLayout>
#include <QGridLayout>
#include <QTextEdit>
#include <QLineEdit>
#include <QTimer>
#include <QEventLoop>
#include <QToolButton>
#include <QFileDialog>
#include <QApplication>
#include <QScreen>

#include "DefaultCard.h"
#include "CardQ_APicture.h"
#include "SpacedRepetitionSystem.h"

class DisplayCardWindow:public QWidget{
    Q_OBJECT
private:
    QPushButton*accept;
    QPushButton*back;
    QLineEdit*Name;
    QTextEdit*Text;
    QTextEdit*Question;
    QTextEdit*Answer;
    QLabel*Picture;

    Deck*deck;
    Card*card;
    std::string prevName;

    void delay();
public:
    explicit DisplayCardWindow(QWidget*wgt=nullptr,Card*card=
nullptr,Deck*deck=nullptr);
signals:

```



```

        void returnBack();
public slots:
    void backClicked();
    void saveName(const QString&);
    void saveText();
    void saveQuestion();
    void saveAnswer();
    void savePicture();
    void accepted();
};

```

### **DisplayCardWindow.cpp**

```
#include "DisplayCardWindow.h"
```

```

DisplayCardWindow::DisplayCardWindow(QWidget*wgt,
Card*crd, Deck*dck) : QWidget(wgt)
{
    card=crd;
    deck=dck;
    prevName=card->getName();
    std::string type=card->getType();

    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    back = new QPushButton("Вернуться");
    back-
>setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    back->setMaximumSize(0.104*width, 0.046*height);

    accept=new QPushButton("Подтвердить");
    accept-
>setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    accept->setMaximumSize(0.104*width, 0.046*height);

    QString string = string.fromStdString(card->getName());

    Name = new QLineEdit(); //создание поле ввода
    Name->setText(string);
    Name-
>setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    Name->setMaximumSize(0.416*width, 0.046*height);
    Name->setStyleSheet(QString("font-size: %1px").arg(16));

    auto*NameLbl=new QLabel("Название:");
    NameLbl->setMaximumSize(0.104*width, 0.046*height);
    NameLbl->setStyleSheet(QString("font-size: %1px").arg(16));

```

```

NameLbl->setBuddy (Name);

auto*grdLayout=new QGridLayout;
grdLayout->setAlignment (Qt::AlignHCenter);
grdLayout->setRowMinimumHeight (0,0.046*height);
grdLayout->setRowMinimumHeight (1,0);
grdLayout->setRowMinimumHeight (2,0);
grdLayout->setRowMinimumHeight (3,0);
grdLayout->setRowMinimumHeight (4,0);

grdLayout->setColumnMinimumWidth (0,0.104*width);
grdLayout->setColumnMinimumWidth (1,0);
grdLayout->setColumnMinimumWidth (2,0);

grdLayout->addWidget (NameLbl,0,0);
grdLayout->addWidget (Name,0,1);

if (type=="DefaultCard")
{
    auto *defCard=dynamic_cast<DefaultCard*> (card);
    std::string text=defCard->getText();
    text=Deck::replaceAll (text,"\\[enter\\]", "\\n");
    string=string.fromStdString (text);

    Text=new QTextEdit();
    Text->insertPlainText (string);
    Text-
>setSizePolicy (QSizePolicy::Expanding,QSizePolicy::Expanding);
    Text->setMaximumSize (0.416*width,0.37*height);
    Text->setStyleSheet (QString ("font-size: %1px").arg (16));

    auto*TextLbl=new QLabel ("Содержание:");
    TextLbl->setMaximumSize (0.104*width,0.046*height);
    TextLbl->setStyleSheet (QString ("font-size:
%1px").arg (16));;
    TextLbl->setBuddy (Text);

    grdLayout->setRowMinimumHeight (1,0.046*height);
    grdLayout->setColumnMinimumWidth (1,0.416*width);

    grdLayout->addWidget (TextLbl,1,0);
    grdLayout->addWidget (Text,1,1);

    connect (Text, SIGNAL (textChanged()), SLOT (saveText()));
    //связь изменения текста с сохранением
}
else if (type=="CardQ_A" || type=="CardQ_APicture")

```

```

{
    auto* cardQ_A=dynamic_cast<CardQ_A*>(card);
    std::string question=cardQ_A->getQuestion();
    question=Deck::replaceAll(question,"\\[enter\\]", "\\n");
    string=string.fromStdString(question);

    Question=new QTextEdit();
    Question->insertPlainText(string);
    Question-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Question->setMaximumSize(0.416*width,0.185*height);
    Question->setStyleSheet(QString("font-size:
%1px").arg(16));

    auto*QuestionLbl=new QLabel("Вопрос:");
    QuestionLbl->setMaximumSize(0.104*width,0.046*height);
    QuestionLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
    QuestionLbl->setBuddy(Question);

    std::string answer=cardQ_A->getAnswer();
    answer=Deck::replaceAll(answer,"\\[enter\\]", "\\n");
    string=string.fromStdString(answer);

    Answer=new QTextEdit();
    Answer->insertPlainText(string);
    Answer-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Answer->setMaximumSize(0.416*width,0.185*height);
    Answer->setStyleSheet(QString("font-size:
%1px").arg(16));

    auto*AnswerLbl=new QLabel("Ответ:");
    AnswerLbl->setMaximumSize(0.104*width,0.046*height);
    AnswerLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
    AnswerLbl->setBuddy(Answer);

    grdLayout->setRowMinimumHeight(2,0.046*height);
    grdLayout->setRowMinimumHeight(3,0.046*height);

    grdLayout->addWidget(QuestionLbl,2,0);
    grdLayout->addWidget(Question,2,1);
    grdLayout->addWidget(AnswerLbl,3,0);
    grdLayout->addWidget(Answer,3,1);

```

```

connect(Question, SIGNAL(textChanged()), SLOT(saveQuestion()));
//связь изменения вопроса с сохранением
    connect(Answer,
SIGNAL(textChanged()), SLOT(saveAnswer())); //связь изменения
ответа с сохранением

    if(type=="CardQ_APicture")
    {

auto*cardQ_APicture=dynamic_cast<CardQ_APicture*>(card);
    string=string.fromStdString(cardQ_APicture-
>getPicture());
        QPixmap image(string);

        auto*ChooseFile=new QPushButton;
        ChooseFile-
>setMaximumSize(0.036*width,0.023*height);
        ChooseFile->setText("Выбрать:");

        auto*ChooseFileLbl=new QLabel("Выберите картинку:");
        ChooseFileLbl-
>setMaximumSize(0.104*width,0.046*height);
        ChooseFileLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
        ChooseFileLbl->setBuddy(ChooseFile);

        Picture=new QLabel;
        Picture-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
        Picture->setMaximumSize(0.208*width,0.185*height);
        Picture->setScaledContents(true);
        Picture->setPixmap(image);

        auto*cellGrid=new QHBoxLayout;
        cellGrid->addWidget(ChooseFileLbl);
        cellGrid->addWidget(ChooseFile);

        grdLayout->setRowMinimumHeight(4,0.023*height);
        grdLayout->addLayout(cellGrid,4,0);
        grdLayout->addWidget(Picture,4,1,Qt::AlignHCenter);

        connect(ChooseFile,
SIGNAL(clicked()), SLOT(savePicture())); //связь нажатия кнопки
с вызовом метода savePicture()
    }
}

```

```

    auto*hButtonsLayout=new QHBoxLayout;
    hButtonsLayout->setAlignment(Qt::AlignHCenter);
    hButtonsLayout->setSpacing(0.104*width);
    hButtonsLayout->addWidget(accept);
    hButtonsLayout->addWidget(back);

    auto*vLayout=new QVBoxLayout;
    vLayout->addLayout(grdLayout);
    vLayout->addLayout(hButtonsLayout);

    setLayout(vLayout);

    connect(Name,
    SIGNAL(textChanged(QString)),SLOT(saveName(QString)));
    connect(back, SIGNAL(clicked()),SLOT(backClicked()));
    connect(accept, SIGNAL(clicked()),SLOT(accepted()));
}

void DisplayCardWindow::backClicked()
{
    emit returnBack();
    delay();
    close();
}

void DisplayCardWindow::saveName(const QString& name)
{
    Name->disconnect();
    card->setName(name.toStdString());
    connect(Name,
    SIGNAL(textChanged(QString)),SLOT(saveName(QString)));
}

void DisplayCardWindow::saveText()
{
    auto*text= qobject_cast<QTextEdit*>(sender());
    auto*defCard=dynamic_cast<DefaultCard*>(card);
    std::string txt=text->toPlainText().toStdString();
    txt=Deck::replaceAll(txt,"\n","[enter]");
    defCard->setText(txt);
}

void DisplayCardWindow::saveQuestion()
{
    auto*question= qobject_cast<QTextEdit*>(sender());
    auto*cardQ_A=dynamic_cast<CardQ_A*>(card);
    std::string qst=question->toPlainText().toStdString();

```

```

        qst=Deck::replaceAll(qst, "\n", "[enter]");
        cardQ_A->setQuestion(qst);
    }

void DisplayCardWindow::saveAnswer()
{
    auto*answer= qobject_cast<QTextEdit*>(sender());
    auto*cardQ_A=dynamic_cast<CardQ_A*>(card);
    std::string answ=answer->toPlainText().toStdString();
    answ=Deck::replaceAll(answ, "\n", "[enter]");
    cardQ_A->setAnswer(answ);
}

void DisplayCardWindow::savePicture()
{
    QString
    filename=QFileDialog::getOpenFileName(nullptr, "Выберите
изображение", "", "*.png *.jpg *.jpeg *.bmp"); //поиск картинки
    if(!filename.isEmpty())
    {
        QPixmap image(filename); // получение картинки
        Picture->setPixmap(image); // установка картинки в
QLabel
    }
    auto*cardQ_APicture=dynamic_cast<CardQ_APicture*>(card);
    std::string AddressPicture=filename.toStdString();
    //сохранение пути
    cardQ_APicture->setPicture(AddressPicture);
}

void DisplayCardWindow::accepted()
{
    deck->rewriteInFile();
    emit returnBack();
    delay();
    close();
}

void DisplayCardWindow::delay()
{
    QEventLoop loop;
    QTimer timer;
    timer.setInterval(100); //0.1sec
    connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
    timer.start();
}

```

```

        loop.exec();
    }

```

### **CreatingCardWindow.h**

```

#pragma once
#include <QWidget>
#include <QTextEdit>
#include <QPushButton>
#include <QLineEdit>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QFileDialog>
#include <QToolButton>
#include <QLabel>
#include <QGridLayout>
#include <QFormLayout>
#include <QApplication>
#include <QScreen>
#include <QDate>
#include <QTime>
#include <QTimer>

#include "Session.h"

class CreatingCardWindow: public QWidget{
    Q_OBJECT
private:
    Deck*deck;
    QLineEdit*Name;
    QTextEdit*Text;
    QTextEdit*Question;
    QTextEdit*Answer;
    QLabel*Picture;

    QToolButton*ChooseFile;
    QPushButton*Accept;
    QPushButton*Reject;

    QString AddressPicture;
    QString SavedText;
    QString SavedQuestion;
    QString SavedAnswer;

    void delay();
signals:
    void returnBack();
public:

```

```

        explicit
        CreatingCardWindow(QWidget*wgt=nullptr,Deck*deck=nullptr);
public slots:
    void accepted();
    void rejected();
    void textCard();
    void Q_ACard();
    void pictureChoose();
};

```

### **CreatingCardWindow.cpp**

```

#include "CreatingCardWindow.h"

CreatingCardWindow::CreatingCardWindow(QWidget *wgt,Deck*dck) :
QWidget(wgt)
{
    //получение размеров экрана
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    deck=dck;
    //создание элементов окна
    Accept=new QPushButton("Создать");
    Accept-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    //политика расширения
    Accept->setMaximumSize(0.104*width,0.046*height);
    //установка размера текста

    Reject=new QPushButton("Отменить");
    Reject-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Reject->setMaximumSize(0.104*width,0.046*height);

    Name=new QLineEdit;
    Name-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Name->setMaximumSize(0.416*width,0.046*height);
    Name->setStyleSheet(QString("font-size: %1px").arg(16));

    Text=new QTextEdit;
    Text-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Text->setMaximumSize(0.416*width,0.185*height);
    Text->setStyleSheet(QString("font-size: %1px").arg(16));

```



```

    Question=new QTextEdit;
    Question-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Question->setMaximumSize(0.416*width,0.185*height);
    Question->setStyleSheet(QString("font-size: %1px").arg(16));

    Answer=new QTextEdit;
    Answer-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Answer->setMaximumSize(0.416*width,0.185*height);
    Answer->setStyleSheet(QString("font-size: %1px").arg(16));

    ChooseFile=new QToolButton;
    ChooseFile->setMaximumSize(0.036*width,0.023*height);
    ChooseFile->setText("Выбрать");

    Picture=new QLabel;
    Picture-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Picture->setMaximumSize(0.208*width,0.185*height);
    Picture->setScaledContents(true);    //масштабирование
картинки по размеру

```

```

    auto*NameLbl=new QLabel("Введите название:");
    NameLbl->setMaximumSize(0.104*width,0.046*height);
    NameLbl->setStyleSheet(QString("font-size: %1px").arg(16));
    NameLbl->setBuddy(Name);    //привязка метки к полю

```

```

    auto*TextLbl=new QLabel("Введите текст:");
    TextLbl->setMaximumSize(0.104*width,0.046*height);
    TextLbl->setStyleSheet(QString("font-size: %1px").arg(16));
    TextLbl->setBuddy(Text);

```

```

    auto*QuestionLbl=new QLabel("Введите вопрос:");
    QuestionLbl->setMaximumSize(0.104*width,0.046*height);
    QuestionLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
    QuestionLbl->setBuddy(Question);

```

```

    auto*AnswerLbl=new QLabel("Введите ответ:");
    AnswerLbl->setMaximumSize(0.104*width,0.046*height);
    AnswerLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
    AnswerLbl->setBuddy(Answer);

```

```

    auto*ChooseFileLbl=new QLabel("Выберите картинку:");

```

```

ChooseFileLbl->setMaximumSize(0.104*width,0.046*height);
ChooseFileLbl->setStyleSheet(QString("font-size:
%1px").arg(16));
ChooseFileLbl->setBuddy(ChooseFile);

auto*grdLayout=new QGridLayout; //компоновщик таблицы
grdLayout->setAlignment(Qt::AlignHCenter); //выравнивание
по центру
grdLayout->setRowMinimumHeight(0,0.046*height);
//минимальная высота строки
grdLayout->setRowMinimumHeight(1,0.046*height);
grdLayout->setRowMinimumHeight(2,0.046*height);
grdLayout->setRowMinimumHeight(3,0.046*height);
grdLayout->setRowMinimumHeight(4,0.046*height);

grdLayout->setColumnMinimumWidth(0,0.104*width);
//минимальная ширина колонны
grdLayout->setColumnMinimumWidth(1,0.416*width);

grdLayout->addWidget(NameLbl,0,0);
grdLayout->addWidget(Name,0,1);
grdLayout->addWidget(TextLbl,1,0);
grdLayout->addWidget(Text,1,1);
grdLayout->addWidget(QuestionLbl,2,0);
grdLayout->addWidget(Question,2,1);
grdLayout->addWidget(AnswerLbl,3,0);
grdLayout->addWidget(Answer,3,1);

auto*cellGrid=new QHBoxLayout;
cellGrid->addWidget(ChooseFileLbl);
cellGrid->addWidget(ChooseFile);

grdLayout->addLayout(cellGrid,4,0);
grdLayout->addWidget(Picture,4,1,Qt::AlignHCenter);

auto*hButtonsLayout=new QHBoxLayout;
hButtonsLayout->setAlignment(Qt::AlignHCenter);
hButtonsLayout->setSpacing(0.104*width);
hButtonsLayout->addWidget(Accept);
hButtonsLayout->addWidget(Reject);

auto*vLayout=new QVBoxLayout;
vLayout->addLayout(grdLayout);
vLayout->addLayout(hButtonsLayout);

```

```

        setLayout(vLayout);

        connect(Reject, SIGNAL(clicked()), SLOT(rejected()));
//связь кнопки с выходом
        connect(Accept, SIGNAL(clicked()), SLOT(accepted()));
//связь кнопки с подтверждением
        connect(Text, SIGNAL(textChanged()), SLOT(textCard()));
//связь изменения текста с сохранением
        connect(Question, SIGNAL(textChanged()), SLOT(Q_ACard()));
//связь изменения текста с сохранением
        connect(Answer, SIGNAL(textChanged()), SLOT(Q_ACard()));
//связь изменения текста с сохранением
        connect(ChooseFile,
SIGNAL(clicked()), SLOT(pictureChoose())); //связь кнопки с
выбором картинки
    }

void CreatingCardWindow::rejected()
{
    emit returnBack(); //вызов сигнала перехода на предыдущее
окно
    delay();
    close(); //закрытие окна
}

void CreatingCardWindow::accepted()
{
    QDate Qdate=QDate::currentDate(); //олучение текущей даты
    std::string
date=(Qdate.toString("dd.MM.yyyy")).toStdString();

    QTime Qtime=QTime::currentTime(); //получение текущего
времени
    std::string time=(Qtime.toString("hh:mm:ss")).toStdString();

    std::string inf[9];
    inf[0]=Name->text().toStdString();
    inf[1]=Text->toPlainText().toStdString();
    inf[2]=Question->toPlainText().toStdString();
    inf[3]=Answer->toPlainText().toStdString();
    inf[4]=AddressPicture.toStdString();
    inf[5]=date;
    inf[6]=time;
    inf[7]="0";
    inf[8]="True";
    try

```

```

    {
        deck->createCard(inf, true); //создание карточки
    }
    catch(const MyException&exception)
    {
        exception.show(); //отображения исключения
        deck->rewriteInFile(); //перезапись файла колоды
    }
    catch(...)
    {
        while (QMessageBox::warning(nullptr, "Error", "Unknown
error!", QMessageBox::Apply) !=
            QMessageBox::Apply);
    }

    emit returnBack();
    delay();
    close();
}

void CreatingCardWindow::textCard()
{
    QString text=Text->toPlainText(); //получение текста с поля
    if(!text.isEmpty()) {
        if(SavedQuestion.isEmpty())
        {
            SavedQuestion = Question->toPlainText();
        }
        if( SavedAnswer.isEmpty())
        {
            SavedAnswer = Answer->toPlainText();
        }

        Question->disconnect();
        Question->clear();
        Question->setReadOnly(true);

        Answer->disconnect();
        Answer->clear();
        Answer->setReadOnly(true);

        ChooseFile->setEnabled(false);
    }
    else
    {
        Question->setReadOnly(false);
        Question->setText(SavedQuestion);
    }
}

```

```

        connect (Question, SIGNAL (textChanged()), SLOT (Q_ACard()));

        Answer->setReadOnly(false);
        Answer->setText (SavedAnswer);
        connect (Answer, SIGNAL (textChanged()), SLOT (Q_ACard()));

        ChooseFile->setEnabled(true);

        if (!SavedQuestion.isEmpty()) {
            SavedQuestion = "";
        }
        if (!SavedAnswer.isEmpty()) {
            SavedAnswer="";
        }
    }
}

void CreatingCardWindow::Q_ACard()
{
    QString question=Question->toPlainText();
    QString answer=Answer->toPlainText();
    if (!question.isEmpty() || !answer.isEmpty()) {
        if (SavedText.isEmpty())
        {
            SavedText=Text->toPlainText();
        }
        Text->disconnect(); //убрать связь поля Text() с
изменением текста
        Text->clear();
        Text->setReadOnly(true);
    }
    else
    {
        Text->setReadOnly(false);
        Text->setText (SavedText);
        connect (Text, SIGNAL (textChanged()), SLOT (textCard()));
        if (!SavedText.isEmpty())
        {
            SavedText="";
        }
    }
}

void CreatingCardWindow::pictureChoose()
{
    QString
filename=QFileDialog::getOpenFileName(nullptr, "Выберите

```

```

изображение","", "*.png *.jpg *.jpeg *.bmp"); //выбор картинки
    if(!filename.isEmpty())
    {
        AddressPicture=filename;
        QPixmap image(AddressPicture); //создание изображения
        Picture->setPixmap(image); //установка его в QLabel
    }
}

void CreatingCardWindow::delay()
{
    QEventLoop loop;
    QTimer timer;
    timer.setInterval(100); //0.1sec
    connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
    timer.start(); //запуск таймера
    loop.exec();
}

```

#### **RepetitionWindow.h**

```

#pragma once
#include <QWidget>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QTableWidget>
#include <QPushButton>
#include <QLabel>
#include <QHeaderView>
#include <QStackedWidget>
#include <QApplication>
#include <QEventLoop>
#include <QTimer>

#include "Session.h"
#include "DeckButton.h"
#include "RepetitionProcessWindow.h"

class RepetitionWindow:public QWidget{
    Q_OBJECT
private:
    Session*session;
    QPushButton*back;
    QTableWidget*tbl;
    QHBoxLayout*hLayout;

    QStackedWidget*stack;

```

```

        int curRow=-1;
        int curColumn=0;

        void delay();
signals:
        void returnBack();
public:
        explicit RepetitionWindow(QWidget*wgt=nullptr);
        ~RepetitionWindow(){delete session;}
        void takeFromSession();
        void addDeck(const QString&);
        void renewTbl();
        void intProcess();
public slots:
        void deckClicked();
        void backClicked();
        void changeWindow();
};

```

#### **RepetiitonWindow.cpp**

```
#include "RepetitionWindow.h"
```

```

RepetitionWindow::RepetitionWindow(QWidget *wgt) : QWidget(wgt)
{
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    back=new QPushButton("Вернуться");
    back-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    back->setMaximumSize(0.104*width,0.046*height);

    stack=nullptr;

    tbl=new QTableWidget(0,9);
    tbl->setMaximumSize(0.753*width,0.815*height);
    tbl->horizontalHeader()->setVisible(false);
    tbl->verticalHeader()->setVisible(false);
    tbl->setShowGrid(false);
    tbl->setSelectionMode(QAbstractItemView::NoSelection);
    tbl->setEditTriggers(QAbstractItemView::NoEditTriggers);
    tbl->setFocusPolicy(Qt::NoFocus);
    tbl->setStyleSheet("QTableWidget::item{padding: 10px}");

    auto*vLayoutMain=new QVBoxLayout;
    vLayoutMain->setAlignment(Qt::AlignTop);

```

```

vLayoutMain->addSpacing(0.047*height);
vLayoutMain->setSpacing(0.01*width);
vLayoutMain->addWidget(back);

hLayout=new QHBoxLayout;
hLayout->addLayout(vLayoutMain);
hLayout->addWidget(tbl);
setLayout(hLayout);
try
{
    session=new Session;
    session->initialisation();
    session-
>setActiveDecks(SpacedRepetitionSystem::createActiveDecks(session-
n->getActiveDecks(),session->getDecks()));
    renewTbl();
    takeFromSession();
}
catch( const MyException& exception)
{
    throw;
}
catch(...)
{
    throw;
}

connect(back,SIGNAL(clicked()),SLOT(backClicked()));
}

void RepetitionWindow::addDeck(const QString& string)
{
    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    if(curColumn==0)
    {
        tbl->setRowCount(tbl->rowCount() + 1);
        curRow++;
        tbl->setRowHeight(tbl->rowCount() - 1, 0.139*height);
    }
    if (curColumn == tbl->columnCount())
    {
        tbl->setRowCount(tbl->rowCount() + 1);
        curRow++;
        tbl->setRowHeight(tbl->rowCount() - 1, 0.139*height);
        curColumn = 0;
    }
}

```



```

    }
    tbl->setColumnWidth(curColumn, 0.083*width);

    auto*wgt=new QWidget;

    auto *item = new DeckButton;
    item->setTopic(string);
    item-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);

    auto*label=new QLabel(string);
    label->setAlignment(Qt::AlignHCenter | Qt::AlignBottom);
    label->setBuddy(item);

    auto*vLayout=new QVBoxLayout;
    vLayout->addWidget(item);
    vLayout->addWidget(label);
    wgt->setLayout(vLayout);

    tbl->setCellWidget(curRow, curColumn, wgt);
    curColumn++;

    connect(item, SIGNAL(clicked()),SLOT(deckClicked()));
}

void RepetitionWindow::takeFromSession()
{
    std::map<std::string ,Deck*> Copy=session->getActiveDecks();
    for(const auto &item: Copy)
    {
        std::string str=item.first;
        QString string;
        string=string.fromStdString(str);
        addDeck(string);
    }
}

void RepetitionWindow::renewTbl()
{
    for(int i=0;i<tbl->rowCount();i++)
        for(int j=0;j<tbl->columnCount();j++)
        {
            if(i==curColumn && j==curColumn)
            {
                i=tbl->rowCount();
                break;
            }
        }
}

```

```

        }
        tbl->removeCellWidget(i,j);
    }

    curRow=-1;
    curColumn=0;
    tbl->setRowCount(0);
    tbl->setColumnCount(9);
}

void RepetitionWindow::backClicked()
{
    emit returnBack();
    delay();
    close();
}

void RepetitionWindow::deckClicked()
{
    auto*button= qobject_cast<DeckButton*>(sender());
    std::string str=button->getTopic();

    Deck*deck=session->getActiveDecks()[str];
    std::list<Card*> activeCards=deck->getActiveCards();

    stack=new QStackedWidget;
    stack->setAttribute(Qt::WA_DeleteOnClose);

    Statistics*statistics=session->getStatistics();

    for(const auto& item:activeCards)
    {
        auto*repetitionProcessWindow=new
        RepetitionProcessWindow(nullptr,item,deck,statistics);
        repetitionProcessWindow->
        setAttribute(Qt::WA_DeleteOnClose);
        stack->addWidget(repetitionProcessWindow);

        connect(repetitionProcessWindow,&RepetitionProcessWindow::return
        Back,this,&RepetitionWindow::intProcess);

        connect(repetitionProcessWindow,&RepetitionProcessWindow::nextCa
        rd,this,&RepetitionWindow::changeWindow);
    }
    stack->showFullScreen();
    delay();
}

```

```

        hide();
    }

void RepetitionWindow::changeWindow()
{
    int index=stack->currentIndex()+1;
    if(index>=stack->count())
    {
        intProcess();
        return;
    }

    stack->setCurrentIndex(index);
    qApp->processEvents(QEventLoop::ExcludeUserInputEvents);
}

void RepetitionWindow::intProcess()
{
    session-
>setActiveDecks(SpacedRepetitionSystem::createActiveDecks(session-
n->getActiveDecks(),session->getDecks()));
    renewTbl();
    takeFromSession();
    show();
    delay();
    stack->close();
}

void RepetitionWindow::delay()
{
    QEventLoop loop;
    QTimer timer;
    timer.setInterval(100); //0.1sec
    connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
    timer.start();
    loop.exec();
}

```

### **RepetitionProcessWindow.h**

```

#pragma once

#include <QWidget>
#include <QPushButton>
#include <QPixmap>
#include <QPalette>
#include <QVBoxLayout>
#include <QHBoxLayout>

```

```

#include <QLabel>
#include <QTextEdit>
#include <QLineEdit>
#include <QGridLayout>
#include <QMessageBox>
#include <QEventLoop>
#include <QTimer>
#include <QScreen>
#include <QApplication>

#include "CardQ_APicture.h"
#include "DefaultCard.h"
#include "SpacedRepetitionSystem.h"
#include "Statistics.h"

class RepetitionProcessWindow: public QWidget{
    Q_OBJECT
private:
    Statistics*statistics;
    Deck*deck;
    Card*ActiveCard;
    std::string Type;
    std::string SavedAnswer;

signals:
    void returnBack();
    void nextCard();

public:
    explicit RepetitionProcessWindow(QWidget
*wgt=nullptr,Card*ActiveCard=nullptr,Deck*deck=nullptr,Statistic
s*statistics=nullptr);

public slots:
    void checkClicked();
    void backClicked();
    void saveAnswer();
};

```

#### **RepetitionProcessWindow.cpp**

```

#include "RepetitionProcessWindow.h"

```

```

RepetitionProcessWindow::RepetitionProcessWindow(QWidget*wgt,Car
d*card,Deck*dck,Statistics*stats): QWidget(wgt)
{
    ActiveCard = card;
    Type=card->getType();
    deck=dck;

```

```

statistics=stats;

int height=QApplication::primaryScreen()-
>geometry().height();
int width=QApplication::primaryScreen()->geometry().width();

auto *back = new QPushButton("Вернуться");
back-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
back->setMaximumSize(0.104*width,0.046*height);

auto *check = new QPushButton("Проверить");
check-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
check->setMaximumSize(0.104*width,0.046*height);

QString string = string.fromStdString(ActiveCard-
>getName());

auto *Name = new QLineEdit(string);
Name->setReadOnly(true);
Name-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
Name->setMaximumSize(0.416*width,0.046*height);
Name->setStyleSheet(QString("font-size: %1px").arg(16));

auto*name=new QLabel("Название:");
name->setMaximumSize(0.104*width,0.046*height);
name->setStyleSheet(QString("font-size: %1px").arg(16));
name->setBuddy(Name);

auto*grdLayout=new QGridLayout;
grdLayout->setAlignment(Qt::AlignHCenter);
grdLayout->setRowMinimumHeight(0,0.046*height);
grdLayout->setRowMinimumHeight(1,0);
grdLayout->setRowMinimumHeight(2,0);
grdLayout->setRowMinimumHeight(3,0);

grdLayout->setColumnMinimumWidth(0,0.104*width);
grdLayout->setColumnMinimumWidth(1,0);
grdLayout->setColumnMinimumWidth(2,0);

grdLayout->addWidget(name,0,0);
grdLayout->addWidget(Name,0,1);

```

```

if (Type=="DefaultCard")
{
    auto *defCard=dynamic_cast<DefaultCard*>(ActiveCard);
    std::string txt=defCard->getText();
    txt=Deck::replaceAll(txt,"\\[enter\\]", "\\n");
    string=string.fromStdString(txt);

    auto*Text=new QTextEdit();
    Text->setText(string);
    Text->setReadOnly(true);
    Text-
>setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    Text->setMaximumSize(0.416*width, 0.37*height);
    Text->setStyleSheet(QString("font-size: %1px").arg(16));

    auto*text=new QLabel("Содержание:");
    text->setMaximumSize(0.104*width, 0.046*height);
    text->setStyleSheet(QString("font-size:
%1px").arg(16));;
    text->setBuddy(Text);

    grdLayout->setRowMinimumHeight(1, 0.046*height);
    grdLayout->setColumnMinimumWidth(1, 0.416*width);

    grdLayout->addWidget(text, 1, 0);
    grdLayout->addWidget(Text, 1, 1);

}
else if (Type=="CardQ_A" || Type=="CardQ_APicture")
{
    auto* cardQ_A=dynamic_cast<CardQ_A*>(ActiveCard);
    std::string qst=cardQ_A->getQuestion();
    qst=Deck::replaceAll(qst,"\\[enter\\]", "\\n");
    string=string.fromStdString(qst);

    auto*Question=new QTextEdit();
    Question->setText(string);
    Question->setReadOnly(true);
    Question-
>setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    Question->setMaximumSize(0.416*width, 0.185*height);
    Question->setStyleSheet(QString("font-size:
%1px").arg(16));;

    auto*question=new QLabel("Вопрос:");
    question->setMaximumSize(0.104*width, 0.046*height);
    question->setStyleSheet(QString("font-size:

```

```

%1px").arg(16));
    question->setBuddy(Question);

    auto*InputAnswer=new QTextEdit;
    InputAnswer-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    InputAnswer->setMaximumSize(0.416*width,0.185*height);
    InputAnswer->setStyleSheet(QString("font-size:
%1px").arg(16));

    auto*Answer=new QLabel("Введите ответ:");
    Answer->setMaximumSize(0.104*width,0.046*height);
    Answer->setStyleSheet(QString("font-size:
%1px").arg(16));
    Answer->setBuddy(InputAnswer);

    grdLayout->setRowMinimumHeight(2,0.046*height);
    grdLayout->setRowMinimumHeight(3,0.046*height);

    grdLayout->addWidget(question,2,0);
    grdLayout->addWidget(Question,2,1);
    grdLayout->addWidget(Answer,3,0);
    grdLayout->addWidget(InputAnswer,3,1);

    connect(InputAnswer,
SIGNAL(textChanged()),SLOT(saveAnswer()));

    if(Type=="CardQ_APicture")
    {

auto*cardQ_APicture=dynamic_cast<CardQ_APicture*>(ActiveCard);
    string=string.fromStdString(cardQ_APicture-
>getPicture());
    QPixmap image(string);
    auto*Picture=new QLabel;
    Picture-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    Picture->setMaximumSize(0.208*width,0.185*height);
    Picture->setScaledContents(true);
    Picture->setPixmap(image);

    grdLayout->setColumnMinimumWidth(2,0.208*width);

    grdLayout->addWidget(Picture,2,2);

```

```

        }
    }

    connect(back, SIGNAL(clicked()), SLOT(backClicked()));
    connect(check, SIGNAL(clicked()), SLOT(checkClicked()));

    auto*hButtonsLayout=new QHBoxLayout;
    hButtonsLayout->setAlignment(Qt::AlignHCenter);
    hButtonsLayout->setSpacing(0.104*width);
    hButtonsLayout->addWidget(check);
    hButtonsLayout->addWidget(back);

    auto*vLayout=new QVBoxLayout;
    vLayout->addLayout(grdLayout);
    vLayout->addLayout(hButtonsLayout);

    setLayout(vLayout);
}

void RepetitionProcessWindow::backClicked()
{
    emit returnBack();
    hide();
}

void RepetitionProcessWindow::checkClicked()
{
    bool right;
    if(Type=="CardQ_A" || Type=="CardQ_APicture")
    {
        auto *cardQ_A = dynamic_cast<CardQ_A *>(ActiveCard);
        statistics->setAmountAnswers(statistics->getAmountAnswers() + 1);
        if (SavedAnswer == cardQ_A->getAnswer())
        {
            right=true;
            while (QMessageBox::information(nullptr, "Info",
"The correct answer!", QMessageBox::Apply) !=
                QMessageBox::Apply);
            statistics->setAmountRightAnswers(statistics->getAmountRightAnswers() + 1);
        }
        else
        {
            right=false;
            while (QMessageBox::information(nullptr, "Info",
"Mistake!", QMessageBox::Apply) != QMessageBox::Apply);
        }
    }
}

```



```

        }
    }
    else

        right=true;

        SpacedRepetitionSystem::nextDateOfRepetition(ActiveCard,
right);
        deck->rewriteInFile();
        if(statistics->getAmountAnswers())
            statistics->setPercentage((float)statistics->
getAmountRightAnswers()/(float)statistics->getAmountAnswers());
        statistics->rewriteInFile();

        emit nextCard();
        close();
    }

void RepetitionProcessWindow::saveAnswer()
{
    auto*Text= qobject_cast<QTextEdit*>(sender());
    SavedAnswer=Text->toPlainText().toString();
    SavedAnswer=Deck::replaceAll(SavedAnswer,"\\n","[enter]");
}

```

### **StatisticsWindow.h**

```

#pragma once

#include <QApplication>
#include <QWidget>
#include <QPushButton>
#include <QLabel>
#include <QGridLayout>
#include <QScreen>
#include <QVBoxLayout>
#include <QEventLoop>
#include <QTimer>
#include <QMessageBox>

#include "Statistics.h"

class StatisticsWindow:public QWidget{
    Q_OBJECT
private:
    Statistics*statistics;
    void delay();

public:

```

```

        explicit StatisticsWindow(QWidget* wgt=nullptr);
        ~StatisticsWindow() {delete statistics;}
signals:
    void returnBack();
public slots:
    void backClicked();
};

```

### **StatisticsWindow.cpp**

```
#include "StatisticsWindow.h"
```

```

StatisticsWindow::StatisticsWindow(QWidget *wgt): QWidget(wgt)
{
    try
    {
        statistics = new Statistics;
        statistics->initialisation();
    }
    catch (const MyException&exception)
    {
        statistics->rewriteInFile();
        throw;
    }

    int height=QApplication::primaryScreen()-
>geometry().height();
    int width=QApplication::primaryScreen()->geometry().width();

    auto*back=new QPushButton("Вернуться");
    back-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    back->setMaximumSize(0.104*width,0.046*height);

    QString
string=string.fromStdString(std::to_string(statistics-
>getAmountDecks()));
    auto*AmountDecks=new QLabel(string);
    AmountDecks->setAlignment(Qt::AlignCenter);
    AmountDecks->setMaximumSize(0.104*width,0.046*height);
    AmountDecks->setStyleSheet(QString("font-size:
%1px").arg(16));

    string=string.fromStdString(std::to_string(statistics-
>getAmountCards()));
    auto*AmountCards=new QLabel(string);
    AmountCards->setAlignment(Qt::AlignCenter);
    AmountDecks->setMaximumSize(0.104*width,0.046*height);

```

```

    AmountCards->setStyleSheet(QString("font-size:
%1px").arg(16));

    string=string.fromStdString(std::to_string(statistics-
>getAmountAnswers()));
    auto*AmountAnswers=new QLabel(string);
    AmountAnswers->setAlignment(Qt::AlignCenter);
    AmountAnswers->setMaximumSize(0.104*width,0.046*height);
    AmountAnswers->setStyleSheet(QString("font-size:
%1px").arg(16));

    string=string.fromStdString(std::to_string(statistics-
>getAmountRightAnswers()));
    auto*AmountRight=new QLabel(string);
    AmountRight->setAlignment(Qt::AlignCenter);
    AmountRight->setMaximumSize(0.104*width,0.046*height);
    AmountRight->setStyleSheet(QString("font-size:
%1px").arg(16));

    string=string.fromStdString(std::to_string(statistics-
>getPercentage()*100));
    auto*Percentage=new QLabel(string+" %");
    Percentage->setAlignment(Qt::AlignCenter);
    Percentage->setMaximumSize(0.104*width,0.046*height);
    Percentage->setStyleSheet(QString("font-size:
%1px").arg(16));

    auto*decks=new QLabel("Количество колод:");
    decks-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    decks->setMaximumSize(0.150*width,0.07*height);
    decks->setStyleSheet(QString("font-size: %1px").arg(16));
    decks->setBuddy(AmountDecks); //связь метки с меткой

    auto*cards=new QLabel("Количество карточек:");
    cards-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    cards->setMaximumSize(0.150*width,0.07*height);
    cards->setStyleSheet(QString("font-size: %1px").arg(16));
    cards->setBuddy(AmountCards);

    auto*answers=new QLabel("Количество ответов:");
    answers-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    answers->setMaximumSize(0.150*width,0.07*height);
    answers->setStyleSheet(QString("font-size: %1px").arg(16));
    answers->setBuddy(AmountAnswers);

```

```

    auto*right=new QLabel("Правильные ответы:");
    right-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    right->setMaximumSize(0.150*width,0.07*height);
    right->setStyleSheet(QString("font-size: %1px").arg(16));
    right->setBuddy(AmountRight);

    auto*percentage=new QLabel("Процент правильных ответов:");
    percentage-
>setSizePolicy(QSizePolicy::Expanding,QSizePolicy::Expanding);
    percentage->setMaximumSize(0.150*width,0.07*height);
    percentage->setStyleSheet(QString("font-size:
%1px").arg(16));
    percentage->setBuddy(Percentage);

    auto*grdLayout=new QGridLayout;
    grdLayout->setAlignment(Qt::AlignHCenter);
    grdLayout->setRowMinimumHeight(0,0.046*height);
    grdLayout->setRowMinimumHeight(1,0.046*height);
    grdLayout->setRowMinimumHeight(2,0.046*height);
    grdLayout->setRowMinimumHeight(3,0.046*height);
    grdLayout->setRowMinimumHeight(4,0.046*height);

    grdLayout->setColumnMinimumWidth(0,0.104*width);
    grdLayout->setColumnMinimumWidth(1,0.104*width);

    grdLayout->addWidget(decks,0,0);
    grdLayout->addWidget(AmountDecks,0,1);
    grdLayout->addWidget(cards,1,0);
    grdLayout->addWidget(AmountCards,1,1);
    grdLayout->addWidget(answers,2,0);
    grdLayout->addWidget(AmountAnswers,2,1);
    grdLayout->addWidget(right,3,0);
    grdLayout->addWidget(AmountRight,3,1);
    grdLayout->addWidget(percentage,4,0);
    grdLayout->addWidget(Percentage,4,1);

    auto*hButtonLayout=new QHBoxLayout;
    hButtonLayout->addWidget(back,Qt::AlignHCenter);

    auto*vLayout=new QVBoxLayout;
    vLayout->setAlignment(Qt::AlignHCenter);

    vLayout->addLayout(grdLayout);
    vLayout->addSpacing(0.052*height);

```

```

        vLayout->addLayout(hButtonLayout);

        setLayout(vLayout);

        connect(back, SIGNAL(clicked()), SLOT(backClicked()));
    }

void StatisticsWindow::backClicked()
{
    emit returnBack();
    delay();
    close();
}

void StatisticsWindow::delay()
{
    QEventLoop loop;
    QTimer timer;
    timer.setInterval(100);
    connect (&timer, &QTimer::timeout, &loop,
&QEventLoop::quit);
    timer.start();
    loop.exec();
}

```

#### **DeckButton.h**

```

#pragma once
#include <QPushButton>

class DeckButton:public QPushButton {
    Q_OBJECT
private:
    std::string Topic;
public:
    explicit DeckButton(QWidget *wgt = nullptr):
        QPushButton(wgt){};
    std::string getTopic();
    void setTopic(const QString&);
};

```

#### **DeckButton.cpp**

```

#include "DeckButton.h"

std::string DeckButton::getTopic()
{
    return Topic;
}

```

```

}

void DeckButton::setTopic(const QString& string)
{
    Topic=string.toStdString();
}

```

### **InputNameWindow.h**

```

#pragma once
#include <QDialog>
#include <QWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
#include <QString>

class InputNameWindow: public QDialog{
    Q_OBJECT
private:
    QLineEdit* line;
    QString str;
public:
    explicit InputNameWindow(QWidget* wgt=nullptr);
    QString getTopic();
public slots:
    void importText();
};

```

### **InputNameWindow.cpp**

```

#include "InputNameWindow.h"

InputNameWindow::InputNameWindow(QWidget *wgt): QDialog(wgt)
{
    auto *label=new QLabel();
    label->setText("Введите название:");

    line=new QLineEdit;
    label->setBuddy(line);

    auto* Accept=new QPushButton("Принять");
    auto* Decline=new QPushButton("Отклонить");

    auto* hLayout=new QHBoxLayout;
    hLayout->setSpacing(20);
    hLayout->addWidget(Accept);
    hLayout->addWidget(Decline);
}

```

```

        auto* vLayout=new QVBoxLayout;
        vLayout->addStretch();
        vLayout->addWidget(label);
        vLayout->addWidget(line);
        vLayout->addLayout(hLayout);
        vLayout->addStretch();

        setLayout(vLayout);

        connect(Accept, SIGNAL(clicked()),SLOT(importText()));
        //связь нажатия на кнопки с получением текста
        connect(Accept, SIGNAL(clicked()), SLOT(accept()));
        connect(Decline, SIGNAL(clicked()), SLOT(reject()));
    }

    void InputNameWindow::importText()
    {
        str=line->text();
    }

    QString InputNameWindow::getTopic()
    {
        return str;
    }

```

### **WarningWindow.h**

```

#include <QDialog>
#include <QWidget>
#include <QLabel>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QPushButton>

class WarningWindow: public QDialog{
public:
    explicit WarningWindow(QWidget*pwgt=nullptr);
};

```

### **WarningWindow.cpp**

```

#include "WarningWindow.h"

WarningWindow::WarningWindow(QWidget *pwgt): QDialog(pwgt)
{
    auto* label=new QLabel;
    label->setText("Are u sure to exit?");
    auto* Accept=new QPushButton("Yes");

```

```

auto* Reject= new QPushButton("No");

auto * hLayout=new QHBoxLayout();
hLayout->setSpacing(20);
hLayout->addWidget(Accept);
hLayout->addWidget(Reject);


auto* vLayout=new QVBoxLayout();
vLayout->addStretch();
vLayout->addWidget(label);
vLayout->addLayout(hLayout);
vLayout->addStretch();

setLayout(vLayout);

connect(Accept, SIGNAL(clicked()),SLOT(accept()));
connect(Reject, SIGNAL(clicked()),SLOT(reject()));

}

```