Valg af biblioteker til karaoke systemet

"*QGridLayout*" [1]er en layout manager klasse i Qt, det bruges til at opdele vinduet i rækker og kolonner for præcision. Det kan bl.a. bruges til justering af widgets' størrelse og position, dette er en stor hjælp til front-end, for bedre layout og overskuelighed.

```
QGridLayout *gridLayout = new QGridLayout(this);    // Opretter et QGridLayout og knytter det til det nuværende objekt (this)

// Tilføjer et QLabel (ui->label) til række 0, kolonne 0, der strækker sig over 1 række og 2 kolonner, og centrerer det
gridLayout->addWidget(ui->label, 0, 0, 1, 2, Qt::AlignCenter);
gridLayout->addWidget(ui->Song_1, 1, 1);    // Tilføjer et widget (ui->Song_1) til række 1, kolonne 1
gridLayout->addWidget(ui->Song_2, 1, 0);    // Tilføjer et widget (ui->Song_2) til række 1, kolonne 0

// Tilføjer en QPushButton (ui->pushButton) til række 2, kolonne 0, der strækker sig over 1 række og 2 kolonner, og centrerer det
gridLayout->addWidget(ui->pushButton, 2, 0, 1, 2, Qt::AlignCenter);
```

Figur 1: Kodeudsnit for brug af QGridLayout

"*QFile*"[2] er en klasse i Qt, der giver funtionalitet til håndtering af filoperationer i c++ applikationer. Den fungerer som et grænseoverfladeobjekt mellem applikationen og filsystemet. Med QFile kan du udføre almindelige operationer som at åbne, lukke, læse og skrive til filer.

Generelt bruges QFile til at repræsenterer en fil på filsystemet. I vores tilfælde bliver QFile brugt til at håndterer en tekstfil, der indeholder sangteksterne. Først opretter vi et QFile objekt ved at give filstien som parameter. Derefter åbner vi filen læse og tekstilstand vha. "open" metoden.

Når filen åbnes, er der mulighed for at bruge forskellige metoder fra QFile til at læse eller skrive data. Vi bruger readline metoden sammen med et QTextStream objekt til at læse linjerne fra filen og gemme dem i en liste. Dette giver mulighed for dynamisk at indlæse sangteksterne fra filerne og integrere dem i vores applikationslogik.

"*QTextStream*"[3] er en klasse i Qt, der giver faciliteter til at læse og skrive tekst fra forskellige kilder, såsom i vores tilfælde "QString". Det fungerer som en strøm, der tillader håndtering af tekstbaserede data.

For læsning bruger QTextStream streamingoperatorer som "<<" og ">>" til at læse og skrive ord, linjer og tal. Dette gør det nemt at interagere med data i tekstformat. Desuden understøtter QTextStream formateringsoptioner til felt-padding, justering og formatering af tal

Lige i vores specifikke situation bruges QTextStream til at læse sangtekster fra en tekstfil. Det åbner en fil ved hjælp af QFile objektet og forbinder derefter QTextStream til filen for at lette indlæsningen af tektsen linje for linje. Dette er nyttigt, da vin arbejder med eksterne tekstfiler som i vores tilfælde er fra databasen.

---

[1] QGridLayout
[2] QFile
[3] QTextStream

”*QDebug*”[4] er en facilitet i Qt, der bruges til at udskrive fejlmeddelelser og anden information på konsollen under kørsel af applikationen. Det er praktisk værktøj til at lette fejlfinding og overvågning af programadfærd.

Generelt bruges QDebug til at udskrive beskeder, der hjælper udviklere med at forstå, hvad der sker i programmet på et givet tidspunkt. I fores tilfælde bruges QDebug til at producere output, der hjælper med at følge programmets flow og identificere eventuelle problemer.

Vi bruger QDebug funktionen til at skrive beskeder til konsollen. Disse beskeder kan indeholde variabelværdier, statusmeddelelser eller anden relevant information. For eksempel kan vi bruge QDebug til at få besked om sangteksterne blev indlæst fra filerne som de skulle.

Dette hjælper os med at identificerer hvor vi er i programmet samt hjælpe os med at identificere potentielle problemer ved at gøre fejlfinding mere effektiv.

```
void Song1::loadLyrics(const QString &filePath) {
    QFile file(filePath);
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        qDebug() << "Cannot open the file for reading:" << filePath;
        return;
    }

    QTextStream in(&file);
    while (!in.atEnd()) {
        ghostLyrics.append(in.readLine());
    }
    file.close();


    if (!ghostLyrics.isEmpty()) {
        qDebug() << "Lyrics loaded successfully!";
        startLyricsDisplay();
    }
    else {
        qDebug() << "No lyrics loaded!";
    }
}
```

Figur 2: Kodeudsnit for brug af QFile, QTextStream og QDebug

”*QLabel*”[5] er en GUI widget i Qt, der bruges til at vise tekst eller billeder. Det fungerer som et passivt displayområde og tillader visning af statisk tekst eller billeder på brugergrænsefladen.

Generelt bruges QLabel til at præsentere information eller meddelelser for brugeren i en Qt-applikation. I vores tilfælde bliver QLabel anvendt til at vise tekst på skærmen. Det bliver bl.a. brugt til at vise vores sangtekster på skærmen og nedtællingen under afspilning af en sang.

[4] QDebug
[5] QLabel

QLabels udseende og indhold kan tilpasses vha. forskellige metoder og egenskaber. I vores tilfælde ser vi brugen af formateret tekst vha. HTML-tags såsom , til at ændre tekstens farve, stil og farve. Et eksempel på brug af QLabel, hvor vi så sætter farve på en tekst kan ses på figur 3

```
ui->countdownLabel->setText(QString("<span style='color: darkblue;'>%1</span>").arg(count_down_V));
```

Figur 3: Kodeudsnit for brug af QLabel

"QVBoxLayout"[6] er en layout-manager i Qt, der organiserer widgetter i en lodret kolonne. Dette etyder, at elle tilføjede widgetter placeres under hinanden i en enkelt kolonne hvilket skaber en lineær struktur. Dette layout er nyttigt, når du ønsker at have en ordnet sekvens af widgetter fra top til bund.

I vores tilfælde bliver QVBoxLayout brugt til at definere layoutstrukturen for vores vindue. Derudover Implementerer vi QHBoxLayout også kaldet "bottomLayout" til at placerer nogle knapper vandret i bunden. Der tilføjes strækfaktorer mellem knapperne for at opnå den ønskede positionering.

```
// Tilføj din countdownLabel til mainLayout
mainLayout->addWidget(ui->countdownLabel);

// Opret "Back" knappen og tilføj den tsil bottomLayout
QPushButton *on_pushButton_clicked = new QPushButton("Back", this);
bottomLayout->addWidget(on_pushButton_clicked);
connect(on_pushButton_clicked, &QPushButton::clicked, this, &Song1::on_pushButton_clicked);

// Tilføj en strækfaktor mellem "Back" og "Continue" knapperne for at skubbe dem til hver sin side
// Tilføj en strækfaktor mellem "Back" og "Continue" knapperne for at skubbe dem til hver sin side
bottomLayout->addStretch(1);

// Konfigurer "Continue" knappen og tilføj den til bottomLayout
ui->pushButton_2->setStyleSheet("QPushButton { font-size: 18pt; }");
ui->pushButton_2->setText("Continue");
bottomLayout->addWidget(ui->pushButton_2);
connect(ui->pushButton_2, &QPushButton::clicked, this, &Song1::on_pushButton_2_clicked);

// Tilføj bottomLayout til bunden af mainLayout
mainLayout->addLayout(bottomLayout);

// Sæt mainLayout som layout for denne dialog
setLayout(mainLayout);
```

Figur 4: Kodeudsnit for brug af QVBoxLayout

"QPushButton"[7] er en widget i Qt, der repræsenterer en knap, som brugere kan interagere med ved at klikke på den. Denne klasse giver mulighed for oprettelse af knapper med tekst eller ikoner og er en essentiel komponent i opbygning af brugergrænseflader i Qt-applikationer.

I vores implementering af QPushButton bruges det til at oprette adskillige knapper, f.eks. en knap til vinduet for sanglisten fra menuen af. Disse knapper er en del af brugergrænsefladen og bruges til at udføre specifikke handlinger ved at reagere på brugerens interaktion. For at komme tilbage til menuen forbinder vi et slot funktion, kaldet mainmenu. Dette betyder, at når knappen klikkes, udføres

[6] QVBoxLayout
[7] QPushButton

”mainmenu” funktionen, hvilket resulterer i nedlukningen af nuværende vindue og i stedet vender brugeren tilbage til hovedmenuen.

Så QPushButton fungerer som et vigtigt element i brugergrænsefladen, der letter navigationen mellem forskellige sektioner af vores Qt-applikation. Knappen er designet til at være let forståelig og klikbar, hvilket gør det nemt for brugeren at interagere med og styre applikationen.

```
connect(ui->pushButton, &QPushButton::clicked, this, &SongList::on_pushButton_clicked);
```

```
void SongList::on_pushButton_clicked()
{
  if(!mainmenu){

      mainmenu = new MainWindow(this);
  }
  mainmenu->show();
  hide();
}
```

Figur 5: Kodeudsnit for brug af QPushButton

”*Algorithm*”[8] er c++’s bibliotek er en del af Standard Template Library (STL) og indeholder en samling af generiske algoritmer, der kan anvendes på forskellige datastruktur. Det er designet til at gøre det lettere for udviklere at udføre fælles operationer som sortering, manipulation og søgning.

I vores implementering benytter biblioteket std::sort funktionen til at sortere data i faldende rækkefølge baseret på spillernes scores. Dette muliggør en nem og effektiv måde at præsentere highscores på en nem og overskuelig måde, idet de bedste scores bliver visualiseret og vist først.

```
// Sorter highscores i faldende rækkefølge baseret på score
std::sort(combinedHighscores.begin(), combinedHighscores.end(), [](const QPair<QString, int>& a, const QPair<QString, int>& b) {
    return a.second > b.second;
});
```

Figur 6: Kodeudsnit for brug af Algorithm

”*QString*”[9] QString er en klasse i Qt, der håndterer tekststrenge. Den tilbyder en bred mulighed af funktioner til manipulation og repræsentation af tekst. Generelt bruges QString i Qt applikationer til at håndtere tekstbaserede data.

I vores implementering anvendes QString i forskellige sammenhænge. F.eks. bruges det til at repræsentere filstier, da det konveterer en normal c-string til QString vha. ”*QString::fromStdString*()”.

[8] Algorithm
[9] QString

Derudover bruges QString til at oprette formaterede tekststrengbeskeder, især når der er indlæsning af en sangtekst fra en fil. De indlæste linjer konverteres til QString og formateres til at indholde HTML lignende kode, for at vise det som ønsket i brugergrænsefladen.

```
if (!soundEffect) {
    std::string fp = "./../../shared_cache/"+ song.getInstrumentalFile();
    QString qstring_fp = QString::fromStdString(fp);
    // Load the lyrics from the file at the start
    soundEffect = new QSound(qstring_fp, this);
    soundEffect->play();
}
```

Figur 7: Kodeudsnit for brug af QString

"*QTimer*"[10] er en klasse i Qt, der muliggør håndtering af gentagne tidssignaler i Qt applikationer. Den fungerer ved at udløse et signal med jævne mellemrum, og dette signal er forbundet til en bestemt handling, ofte udført gennem et såkaldt "slot" i Qt sprog.

I vores implementering kan det bla.a ses flere steder f.eks. updateCountdown og startLyricsDisplay, da vi har et objekt i updateCountdown der håndterer nedtællingen, for at brugeren er klar over hvornår det er brugeren skal til at synge. I startLyricsDisplay laves der et objekt der håndterer timingen af sangteksterne, for at få det udprintet i takt med den afspillede sang.

```
void Song1::updateCountdown()
{
    count_down_V--;
    if (count_down_V <= 0) {
        count_down_T->stop();
        ui->countdownLabel->setText("<span style='font-size: 30pt; color: darkblue;'>Ready!</span>");
        QTimer::singleShot(1000, this, &Song1::startLyricsDisplay);
    } else {
        ui->countdownLabel->setText(QString("<span style='color: darkblue;'>%1</span>").arg(count_down_V));
    }
}
```

Figur 8: Kodeudsnit for brug af QTimer

"*QDialog*"[11] er en klasse i Qt, der repræsenterer et dialogvindue. Et dialogvindue er et separat vindue, der bruges til at udføre en specifik opgave eller vise oplysninger inden for en applikation. Det kan indeholde forskellige brugergrænsefladeelementer som knapper, tekstfelter, lister og mere.

I vores implementering kan det bl.a. ses at song1 klassen nedarver fra Dialog og repræsenterer et dialogvindue for indspilning af en sang. QDialog bruges til oprettelsen af et separat vindue, der viser sangtekster og en nedtælling samtidig med at brugeren stadig har mulighed for at interagere med brugergrænsefladen, ved at trykke på "cancel" eller "Continue".

[10] QTimer
[11] QDialog

```
namespace Ui {
class Song1;
}

class Song1 : public QDialog
{
    Q_OBJECT

public:
    explicit Song1(QWidget *parent = nullptr);
    ~Song1();

private slots:
    void on_pushButton_clicked();
    void updateCountdown();
    void on_pushButton_2_clicked();
```

Figur 9: Kodeudsnit af klassen for visning af Dialog

```
// Opret layout
QVBoxLayout *mainLayout = new QVBoxLayout;  // Brug QVBoxLayout for at stacke widgets vertikalt
QHBoxLayout *bottomLayout = new QHBoxLayout;  // Brug QHBoxLayout for knapperne i bunden

// Tilføj din countdownLabel til mainLayout
mainLayout->addWidget(ui->countdownLabel);

// Opret "Back" knappen og tilføj den tsil bottomLayout
QPushButton *on_pushButton_clicked = new QPushButton("Back", this);
bottomLayout->addWidget(on_pushButton_clicked);
connect(on_pushButton_clicked, &QPushButton::clicked, this, &Song1::on_pushButton_clicked);

// Tilføj en strækfaktor mellem "Back" og "Continue" knapperne for at skubbe dem til hver sin side
// Tilføj en strækfaktor mellem "Back" og "Continue" knapperne for at skubbe dem til hver sin side
bottomLayout->addStretch(1);

// Konfigurer "Continue" knappen og tilføj den til bottomLayout
ui->pushButton_2->setStyleSheet("QPushButton { font-size: 18pt; }");
ui->pushButton_2->setText("Continue");
bottomLayout->addWidget(ui->pushButton_2);
connect(ui->pushButton_2, &QPushButton::clicked, this, &Song1::on_pushButton_2_clicked);

// Tilføj bottomLayout til bunden af mainLayout
mainLayout->addLayout(bottomLayout);

// Sæt mainLayout som layout for denne dialog
setLayout(mainLayout);
```

Figur 10: Kodeudsnit for brug af QDialog

Det observeres til slut at funktionen ”setLayout” bruger Dialogfunktionaliteten til at indstille layoutet for dialogvinduet baseret på de oprettede widgets, knapper osv.

”*QSound*”[12] er en klasse i Qt, der giver funktionalitet til afspilning af lydfiler. Den bruges til at håndtere lydafspilning i Qt applikationer og understøtter adskillige lydformatter. QSound giver en enkel og bekvem måde at integrere lyd i Qt applikationer uden at skulle bruge nogle overkomplicerede lydbiblioteker.

[12] QSound

I vores implementering bruges QSound til indspilningen af nogle wav-filer der hentes fra serveren ligesom sangteksterne. Først oprettes et objekt af QSound klassen ved at angive stien til lydfilen. Herefter kaldes kommandoer såsom play() metoden som starter indspilningen.

```
if (!soundEffect) {
    std::string fp = "./../../shared_cache/"+ song.getInstrumentalFile();
    QString qstring_fp = QString::fromStdString(fp);
    // Load the lyrics from the file at the start
    soundEffect = new QSound(qstring_fp, this);
    soundEffect->play();
}
```

Figur 11: Kodeudsnit for brug af QSound

MainWindow-klassen:

Denne klasse er hovedkomponenten i denne applikation. Den står for at initiere applikationen og præsentere det primær brugerinterface. Herunder indeholder slot (metoder, som reagerer på signaler i Qt-frameworket) til at håndtering af brugerinteraktioner, hvilket kunne være vælge en sang eller se highscores. Her anvendes der Qt´s signal- og slot mekanisme til at kommunikere med andre komponenter i systemet.

HighScore-klassen:

Denne klasse står som ansvarlig for at præsentere highscore information til brugeren, herunder indeholder der en metode til at vise highscores, hvor der blevet foretaget en sortering af score-data, klassen er centralt forbundet med MainWindow, da brugeren kan tilgå denne menu fra hovedmenuen.

SongList-klassen:

I denne klasse repræsenteres dialogen, hvor brugeren kan vælge mellem forskellige sange, sådan som den er fungere som et mellemled med MainWindow og de specifikke sangdialoger (Song1 og Song2). Den håndterer brugerinputtet og navigere i forhold til den valgte sang fra brugeren.

Song1 og Song2 klasserne:

I følgende klasse indeholder specifikke sangdialoger, som hver indeholder funktionalitet til at indlæse sangtekster fra filer og opdateres ved en nedtælling. Denne klasse er essentielt i

forhold til systemet da den har ansvar for at hente sang data samt synkronisere tekstvisning med lydafspilning. Begge disse klasser har lignende struktur samt funktionalitet.