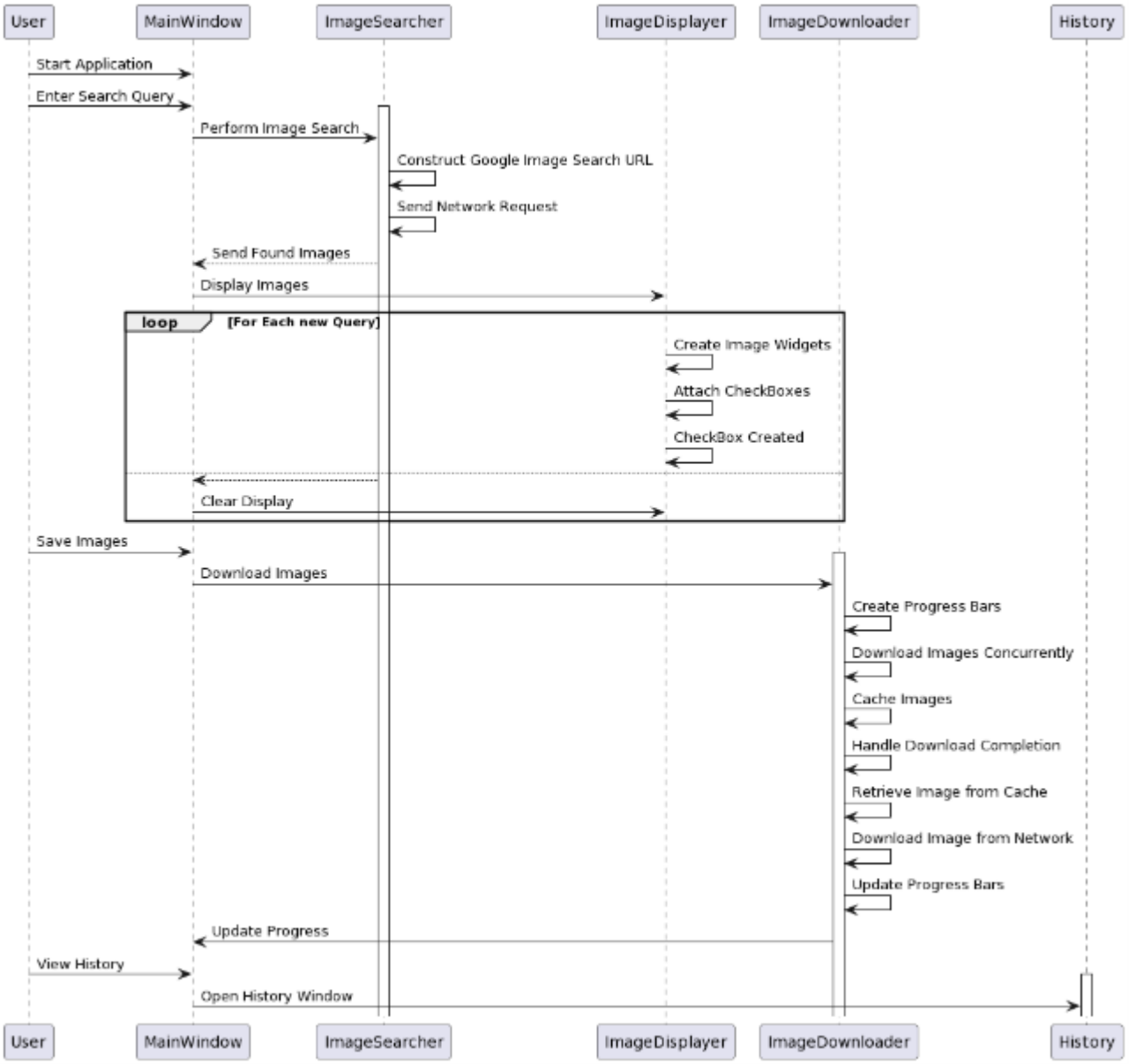
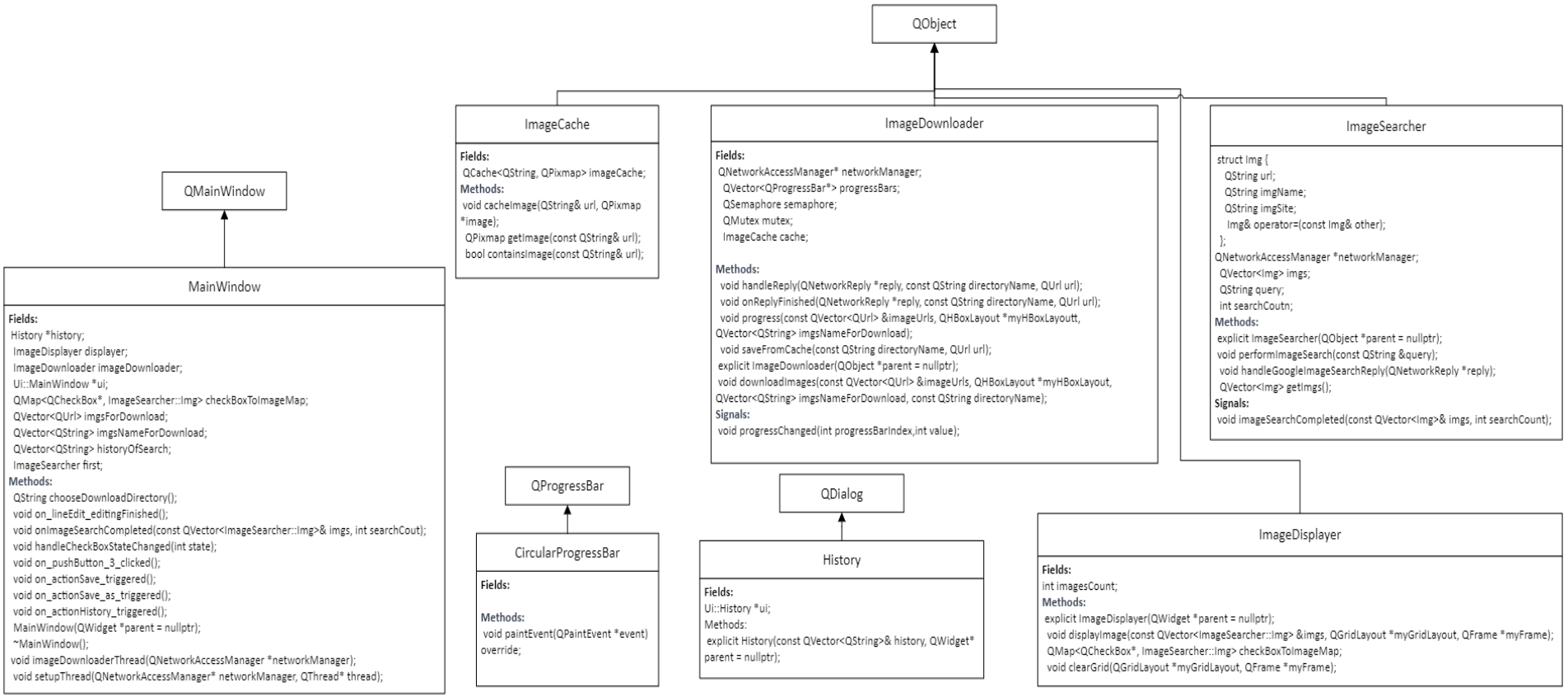
Sequence diagram



Class diagram


QObject
ImageCache
Fields:
QCache<QString, QPixmap> imageCache;
Methods:
void cacheImage(QString& url, QPixmap *image);
QPixmap getImage(const QString& url);
bool containsImage(const QString& url);
ImageDownloader
Fields:
QNetworkAccessManager* networkManager;
QVector<QProgressBar*> progressBars;
QSemaphore semaphore;
QMutex mutex;
ImageCache cache;
Methods:
void handleReply(QNetworkReply *reply, const QString directoryName, QUrl url);
void onReplyFinished(QNetworkReply *reply, const QString directoryName, QUrl url);
void progress(const QVector<QUrl> &imageUrls, QHBoxLayout *myHBoxLayoutt, QVector<QString> imgsNameForDownload);
void saveFromCache(const QString directoryName, QUrl url);
explicit ImageDownloader(QObject *parent = nullptr);
void downloadImages(const QVector<QUrl> &imageUrls, QHBoxLayout *myHBoxLayout, QVector<QString> imgsNameForDownload, const QString directoryName);
Signals:
void progressChanged(int progressBarIndex,int value);
ImageSearcher
struct Img {
QString url;
QString imgName;
QString imgSite;
Img& operator=(const Img& other);
};
QNetworkAccessManager *networkManager;
QVector<Img> imgs;
QString query;
int searchCoutn;
Methods:
explicit ImageSearcher(QObject *parent = nullptr);
void performImageSearch(const QString &query);
void handleGoogleImageSearchReply(QNetworkReply *reply);
QVector<Img> getImgs();
Signals:
void imageSearchCompleted(const QVector<Img>& imgs, int searchCount);
QMainWindow
MainWindow
Fields:
History *history;
ImageDisplayer displayer;
ImageDownloader imageDownloader;
Ui::MainWindow *ui;
QMap<QCheckBox*, ImageSearcher::Img> checkBoxToImageMap;
QVector<QUrl> imgsForDownload;
QVector<QString> imgsNameForDownload;
QVector<QString> historyOfSearch;
ImageSearcher first;
Methods:
QString chooseDownloadDirectory();
void on_lineEdit_editingFinished();
void onImageSearchCompleted(const QVector<ImageSearcher::Img>& imgs, int searchCout);
void handleCheckBoxStateChanged(int state);
void on_pushButton_3_clicked();
void on_actionSave_triggered();
void on_actionSave_as_triggered();
void on_actionHistory_triggered();
MainWindow(QWidget *parent = nullptr);
~MainWindow();
void imageDownloaderThread(QNetworkAccessManager *networkManager);
void setupThread(QNetworkAccessManager* networkManager, QThread* thread);
QProgressBar
CircularProgressBar
Fields:
Methods:
void paintEvent(QPaintEvent *event) override;
QDialog
History
Fields:
Ui::History *ui;
Methods:
explicit History(const QVector<QString>& history, QWidget* parent = nullptr);
ImageDisplayer
Fields:
int imagesCount;
Methods:
explicit ImageDisplayer(QWidget *parent = nullptr);
void displayImage(const QVector<ImageSearcher::Img> &imgs, QGridLayout *myGridLayout, QFrame *myFrame);
QMap<QCheckBox*, ImageSearcher::Img> checkBoxToImageMap;
void clearGrid(QGridLayout *myGridLayout, QFrame *myFrame);

Module: Image Displayer

Description: The "Image Displayer" module is responsible for displaying images on the user interface. It uses Qt for rendering and displaying images retrieved from the web. This module provides functions for dynamically adding and arranging images in a grid layout, along with corresponding labels and checkboxes.

Functions and Methods:

displayImage(const QVector<ImageSearcher::Img> &imgs, QGridLayout *myGridLayout, QFrame *myFrame): This method takes a list of images (imgs) and adds them to the specified grid layout (myGridLayout) within the given frame (myFrame). It also adds labels and checkboxes for each image, creating a visual representation for the images.

clearGrid(QGridLayout *myGridLayout, QFrame *myFrame): This method clears the grid layout (myGridLayout) by removing all added images and their associated components. It resets the frame height within the given frame (myFrame) to its initial value.

Dependencies:

Qt Libraries: This module relies on Qt for various functionalities, such as rendering images, working with grid layouts, and creating user interface components.

Module: Circular Progress Bar

Description: The "Circular Progress Bar" module defines a custom circular progress bar widget for Qt-based applications. This widget inherits from QProgressBar and provides a circular visualization of progress, suitable for tasks such as indicating the progress of a specific operation. The module customizes the appearance of the progress bar to make it circular and visually appealing.

Functions and Methods:

CircularProgressBar(QWidget *parent = nullptr): The constructor initializes the circular progress bar. It sets properties such as text visibility, minimum and maximum values, and the widget's size.

paintEvent(QPaintEvent *event): This method handles the painting of the circular progress bar. It overrides the default paint event to customize the appearance of the widget. The progress is represented as a filled arc within the circular bar.

Dependencies:

Qt Libraries: This module relies on Qt for creating custom graphical user interface components and painting.

Module: History Viewer

Description: The "History Viewer" module defines a dialog widget for viewing and displaying a history of items. This module is designed to create a history viewer dialog within a Qt-based application. It allows users to access and view a list of historical items, such as search queries or recent actions.

Functions and Methods:

History(const QVector<QString> &history, QWidget *parent = nullptr): The constructor initializes the history viewer dialog. It accepts a QVector of QStrings representing the historical items to display and an optional parent widget. Inside the constructor, it sets up the user interface, including the dialog's title and a list widget for displaying the historical items.

~History(): The destructor for the history viewer dialog. It is responsible for cleaning up any allocated resources, including the user interface components.

Dependencies:

Qt Libraries: This module relies on the Qt framework to create graphical user interface elements and manage dialog windows.

Module: Image Cache

Description: The "Image Cache" module is responsible for caching and managing images retrieved from remote URLs within a Qt-based application. It is designed to improve performance and reduce redundant network requests by storing downloaded images locally for quick access.

Functions and Methods:

ImageCache(): The constructor for the Image Cache initializes the cache. This module uses a QCache data structure for storing QPixmap images associated with their respective URLs.

cacheImage(QString& url, QPixmap* image): This method is used to cache an image in the cache. It accepts a reference to a QString representing the image's URL and a pointer to a QPixmap image. The method inserts the image into the cache for future retrieval.

QPixmap getImage(const QString& url): This method retrieves a QPixmap image from the cache based on its URL. It accepts a QString URL as a parameter and returns the corresponding QPixmap image if it exists in the cache. If the image is not found, it returns an empty QPixmap.

bool containsImage(const QString& url): This method checks if an image with a specific URL is present in the cache. It accepts a QString URL and returns true if the image is in the cache and false otherwise.

Dependencies:

Qt Libraries: This module relies on the Qt framework for managing QPixmap images and using QCache to store and retrieve cached images.

Module: Image Downloader

Description: The "Image Downloader" module is responsible for downloading images from provided URLs and saving them to a specified directory. It also offers functionality to display download progress using circular progress bars in the user interface.

Functions and Methods:

ImageDownloader(QObject *parent): The constructor initializes the Image Downloader. It sets up a semaphore to control concurrent downloads and prepares a list of progress bars for displaying download progress.

downloadImages(const QVector<QUrl> &imageUrls, QHBoxLayout *myHBoxLayoutt, QVector<QString> imgsNameForDownload, const QString directoryName): This method initiates the download of images from the specified URLs. It also displays progress bars for each download, accepts a list of image URLs, a horizontal layout for progress bars, and an optional directory for saving images.

handleReply(QNetworkReply *reply, const QString directoryName, QUrl url): This method is responsible for handling network replies and managing concurrent downloads by offloading them to separate threads.

onReplyFinished(QNetworkReply *reply, const QString directoryName, QUrl url): After a download is completed, this method saves the downloaded image to a specified directory and updates the image cache.

saveFromCache(const QString directoryName, QUrl url): This method retrieves images from the cache, which are previously downloaded images, and saves them to the specified directory.

progress(const QVector<QUrl> &imageUrls, QHBoxLayout *myHBoxLayoutt, QVector<QString> imgsNameForDownload): This method sets up circular progress bars for tracking download progress and associates them with labels displaying image names. It also connects progressChanged signals to update the progress bars accordingly.

Dependencies:

Qt Libraries: This module relies on various Qt components, including QNetworkReply, QByteArray, QDir, QFile, QDateTime, QSemaphore, QtConcurrent, QThread, QThreadPool, QRunnable, QFuture, QEventLoop, QProgressBar, QLayout, QLayoutItem, QVBoxLayout, QLabel, QWebEngineView, and QObject.

Interfaces:

None: This module does not expose any external interfaces directly.

Module: Image Searcher

Description: The "Image Searcher" module is responsible for performing image searches on Google based on a provided query. It sends a request to the Google Images search engine, parses the results, and provides a list of image details, including image names, source sites, and URLs.

Functions and Methods:

ImageSearcher(QObject *parent): The constructor initializes the Image Searcher, setting up a network manager for sending requests and preparing variables for query tracking.

performImageSearch(const QString &query): This method initiates an image search for the given query. It constructs a URL for the search, sends a request to Google Images, and specifies the number of results per page and the number of pages to search.

handleGoogleImageSearchReply(QNetworkReply *reply): This method handles the response from the Google Images search. It extracts relevant information from the HTML response, such as image names, source sites, and image URLs, and stores this information in the imgs list. It also emits a signal with the search results and search count.

ImageSearcher::Img& ImageSearcher::Img::operator=(const Img& other): This method is an assignment operator overload for the Img structure. It allows for copying the fields of one Img object into another.

QVector<ImageSearcher::Img> getImgs(): This method retrieves the list of images obtained from the search results.

Dependencies:

Qt Libraries: The module relies on various Qt components, including QNetworkAccessManager, QNetworkRequest, QNetworkReply, QRegularExpression, QString, QXmlStreamReader, QLabel, QVector, and QObject.

Interfaces:

Signal: imageSearchCompleted(QVector<Img> imgs, int searchCount): This signal is emitted when an image search is completed. It provides the list of image details and the current search count.

Module: Main Window

Description: The "Main Window" module is the central component of the application, providing a graphical user interface (GUI) for interacting with various image-related functionalities. This module is the primary user interface for the application and manages communication between the user and other components.

Functions and Methods:

MainWindow(QWidget *parent):

Description: Constructor to create the main application window.

Parameters: QWidget *parent - parent widget (optional).

Functionality: Sets up the main window, connects signals and slots, and initializes the user interface.

on_lineEdit_editingFinished():

Description: Slot function triggered when the user finishes editing the search input.

Functionality: Captures the edited text, adds it to the search history, and initiates an image search for the provided query using the first instance of the ImageSearcher module.

onImageSearchCompleted(const QVector<ImageSearcher::Img>& imgs, int searchCount):

Description: Slot function triggered when an image search is completed.

Parameters:

const QVector<ImageSearcher::Img>& imgs - a list of image details.

int searchCount - the search count.

Functionality: Handles the search results, updates the user interface, and connects checkboxes to the handleCheckBoxStateChanged function.

handleCheckBoxStateChanged(int state):

Description: Slot function triggered when a checkbox state changes.

Parameters: int state - the new state of the checkbox.

Functionality: Handles the checkbox state changes and manages the selection of images for download.

chooseDownloadDirectory():

Description: Opens a dialog for choosing a directory to save downloaded images.

Functionality: Displays a dialog for selecting a download directory and returns the chosen directory's path.

on_pushButton_3_clicked():

Description: Slot function triggered when the user clicks a button to repeat the search.

Functionality: Repeats the image search for the most recent query, disconnects checkboxes from the handleCheckBoxStateChanged function.

on_actionSave_triggered():

Description: Slot function triggered when the user selects the "Save" action.

Functionality: Initiates the image download process and displays download progress for the selected images.

on_actionSave_as_triggered():

Description: Slot function triggered when the user selects the "Save As" action.

Functionality: Allows the user to choose a custom download directory, initiates the image download process, and displays download progress for the selected images.

on_actionHistory_triggered():

Description: Slot function triggered when the user selects the "History" action.

Functionality: Opens a history window to view search history.

Dependencies:

Qt Libraries: This module depends on various Qt components, such as QWidget, QLayoutItem, QLineEdit, QStandardItemModel, and more, for building and managing the user interface.

External Modules: This module interacts with external modules, such as ImageSearcher, ImageDisplayer, ImageDownloader, and History, to perform image searches, display search results, download images, and maintain search history.