"מגשימים" – מוקד כרמיאל	בית ספר "אמירים" כפר ורדים

מסמך עיצוב

MAGSHARE

שם התלמיד: שביט בוריסוב

ת.ז: 318740354

אחראי פרויקט: יואב פוירשטיין

מנטור: מתי פיקוס

תאריך: 12.01.2016

היסטוריית גרסאות המסמך:

תאריך	תיאור
26.12.2015	התחלת העבודה
03.01.2016	ארכיטקטורה

11.01.2016	עיצוב נתונים, ממשק משתמש
12.01.2016	סיום הקובץ
30.04.2016	עדכון בהתאם לפרויקט בפועל
31.05.2016	מספר תוספות

תוכן עניינים

1. מבוא2

2. ארכיטקטורת המערכת2

2.1. מבט על2

2.2. פירוט לכל רכיב2

2.3. דיון לגבי העיצוב הנבחר30

3. עיצוב הנתונים (data)32

4. ממשק משתמש33

5. נספחים35

1. מבוא

מטרת המסמך היא לפרט על מבנה תכנת MagShare. קהל היעד של המסמך הוא עבור מפתחי MagShare.

2. ארכיטקטורת המערכת

2.1. מבט על

Tracker

- מודול תקשורת: מתקשר עם קליינטים. מקבל בקשות לקבצים מקליינטים ומחזיר להם כתובות ל-Peers רלוונטיים. יודע לבדוק מול Peers המבצעים Seeding אם נמצא ברשותם הקובץ המיועד.
- מודול מאגר נתונים: אחראי על שמירת אינפורמציה על Peers פעילים. אינפורמציה זו היא בזיכרון ויש אליה גישה מהירה.
- מודול logging: מאפשר לעקוב באופן נרחב אחר פעולות ה-Tracker.

Client

- מודול תקשורת מול Tracker: שולח בקשות ל-Tracker בכדי לדעת מאיזה Peer להוריד את הקובץ אותו יש להוריד.
- מודול תקשורת מול Peer: שולח בקשות ל-Peer שקיבל מה-Tracker לחלקי הקובץ המיועד, או לחלופין מאזין לבקשות מצד Peers אחרים ושולח להם את חלקי הקובץ אשר נמצא ברשותו.
- מודול ניהול קבצים: אחראי על קריאה מקובץ המיועד לשיתוף וכתיבה לקובץ שייווצר, על פי הגדרות קבצי MSF.
- מודול הגדרות: אחראי על שמירת ההגדרות עבור פעילויותיו של הקליינט.
- מודול UI: ממשק משתמש המאפשר לבחור קבצים, ליצור קבצי MSF על פי ההגדרות המתאימות לפרוטוקול MSP, להתחיל ולעצור הורדות ו-Seeding, וכו'.
- מודול logging: מאפשר לעקוב באופן נרחב אחר פעולות הקליינט.

2.2. פירוט לכל רכיב

להלן הפירוט של המחלקות והפונקציות אשר נמצאות בקליינט וב-Tracker:

Tracker

מודול תקשורת:

ServerThread Class Reference

```
#include <serverthread.h>
```

Public Member Functions

	ServerThread (int descriptor, QObject *parent)
void	run () override
bool	magTrackerQueryParser (QByteArray &data)
QString	magTrackerResponseBuilder ()

Private Attributes

int	descriptor

הסבר: המחלקה אחראית על קבלת בקשות ומתן תשובות בהתאם. האובייקט מקבל בקשה מוצפנת, מפענח אותה, פונה למודול מאגר נתונים (מפורט בהמשך) ומחזיר תשובה בהתאם למודול זה.

מודול מאגר נתונים:

ServerMetaInfo Class Reference

```
#include <servermetainfo.h>
```

Public Member Functions

	ServerMetaInfo ()
void	clear ()
QString	errorString () const
QByteArray	infoHash () const
MetaInfoStruct	metaStruct () const
bool	magTrackerQueryParser (QByteArray &data)

Private Attributes

MetaInfoStruct	metaInfo
QString	errString
QByteArray	content
QByteArray	infoData

הסבר: המחלקה אחראית על פענוח בקשת Peer. האובייקט מפענח את הבקשה ושומר את המידע המפוענח.

Server Class Reference

```
#include <server.h>
```

Public Member Functions

	Server ()

Static Public Member Functions

static void	addClient (const QByteArray &infoHash, const MetaInfoStruct &minfostruct)
static void	removeClient (const QByteArray &infoHash)
static MetaInfoStruct	getClient (const QByteArray &infoHash)
static int	countClients ()

Protected Member Functions

void	incomingConnection (int descriptor) override

Static Private Attributes	
static QMap< QByteArray, MetaInfoStruct >	clients

הסבר: המחלקה אחראית על ניהול מאגר נתוני ה-Peers. האובייקט יודע להוסיף קליינט, להסיר קליינט, להחזיר קליינט לפי בקשת קובץ ולספור את כמות ה-Peers הפעילים ברגע נתון.

Client

מודול תקשורת מול Tracker:

TrackerClient Class Reference

```
#include <trackerclient.h>
```

Public Slots	
void	bytesWritten (qint64 bytes)
void	trackerResponse ()

Signals	
void	connectionError (QString error)
void	failure (const QString &reason)
void	warning (const QString &message)
void	peerListUpdated (const QList< TorrentPeer > &peerList)
void	uploadCountUpdated (qint64 newUploadCount)
void	downloadCountUpdated (qint64 newDownloadCount)
void	stopped ()

Public Member Functions

	TrackerClient (**TorrentClient** *downloader, QObject *parent=0)
void	**start** (const **MetaInfo** &info)
void	**stop** ()
void	**startSeeding** ()

Protected Member Functions

void	**timerEvent** (QTimerEvent *event) Q_DECL_OVERRIDE

Private Slots

void	**fetchPeerList** ()

Private Attributes

QTcpSocket *	**socket**
TorrentClient *	**torrentDownloader**
int	**requestInterval**
int	**requestIntervalTimer**
MetaInfo	**metaInfo**
QByteArray	**trackerId**

QList< TorrentPeer >	peers
qint64	uploadedBytes
qint64	downloadedBytes
qint64	length
QString	uname
QString	pwd
bool	firstTrackerRequest
bool	lastTrackerRequest
bool	firstSeeding

הסבר: המחלקה אחראית על ביצוע תקשורת מול Tracker והעברת נתונים רלוונטיים למודול תקשורת מול Peer ומודול ניהול קבצים. האובייקט יודע לפתוח socket connection ל-Tracker על פורט מוסכם מראש, לבנות ולשלוח בקשה, לקבל תשובה, לפענח תשובה ולסגור את החיבור. לאובייקט יש טיימר אשר "מעיר" אותו כעבור זמן מסוים בכדי לבדוק שינויים מול ה-Tracker.

מודול תקשורת מול Peer:

ConnectionManager Class Reference

```
#include <connectionmanager.h>
```

Public Member Functions

bool	canAddConnection () const
void	addConnection (PeerWireClient *connection)
void	removeConnection (PeerWireClient *connection)
int	maxConnections () const
QByteArray	clientId () const

Static Public Member Functions

static ConnectionManager *	instance ()

Private Attributes

QSet< PeerWireClient * >	connections
QByteArray	id

הסבר: המחלקה אחראית על בדיקת אפשרות לפתיחה, פתיחה והסרה של חיבור ל-Peer. האובייקט מפעיל את אובייקט PeerWireClient (מפורט בהמשך).

PeerWireClient Class Reference

```
#include <peerwireclient.h>
```

Classes

struct	BlockInfo

Public Types

enum	PeerWireStateFlag { ChokingPeer = 0x1, InterestedInPeer = 0x2, ChokedByPeer = 0x4, PeerIsInterested = 0x8 }

Signals

void	infoHashReceived (const QByteArray &infoHash)
void	readyToTransfer ()

void	choked ()
void	unchoked ()
void	interested ()
void	notInterested ()
void	piecesAvailable (const QBitArray &pieces)
void	blockRequested (int pieceIndex, int begin, int length)
void	blockReceived (int pieceIndex, int begin, const QByteArray &data)
void	bytesReceived (qint64 size)

Public Member Functions

	PeerWireClient (const QByteArray &peerId, QObject *parent=0)
void	initialize (const QByteArray &infoHash, int pieceCount)
void	setPeer (TorrentPeer *peer)
TorrentPeer *	peer () const
PeerWireState	peerWireState () const
QBitArray	availablePieces () const
QList< TorrentBlock >	incomingBlocks () const
void	chokePeer ()
void	unchokePeer ()

void	**sendInterested** ()
void	**sendKeepAlive** ()
void	**sendNotInterested** ()
void	**sendPieceNotification** (int piece)
void	**sendPieceList** (const QBitArray &bitField)
void	**requestBlock** (int piece, int offset, int length)
void	**cancelRequest** (int piece, int offset, int length)
void	**sendBlock** (int piece, int offset, const QByteArray &data)
qint64	**writeToSocket** (qint64 bytes)
qint64	**readFromSocket** (qint64 bytes)
qint64	**downloadSpeed** () const
qint64	**uploadSpeed** () const
bool	**canTransferMore** () const
qint64	**bytesAvailable** () const Q_DECL_OVERRIDE
qint64	**socketBytesAvailable** () const
qint64	**socketBytesToWrite** () const
void	**setReadBufferSize** (qint64 size) Q_DECL_OVERRIDE
void	**connectToHost** (const QHostAddress &address, quint16 port,

	OpenMode openMode=ReadWrite) Q_DECL_OVERRIDE
void	**diconnectFromHost** ()

Protected Member Functions

void	**timerEvent** (QTimerEvent *event) Q_DECL_OVERRIDE
qint64	**readData** (char *data, qint64 maxlen) Q_DECL_OVERRIDE
qint64	**readLineData** (char *data, qint64 maxlen) Q_DECL_OVERRIDE
qint64	**writeData** (const char *data, qint64 len) Q_DECL_OVERRIDE

Private Types

enum	**PacketType** {
	ChokePacket = 0, **UnchokePacket** = 1, **InterestedPacket** = 2, **NotInterestedPacket** = 3,
	HavePacket = 4, **BitFieldPacket** = 5, **RequestPacket** = 6, **PiecePacket** = 7,
	CancelPacket = 8
	}

Private Slots

void	**sendHandShake** ()
void	**processIncomingData** ()
void	**socketStateChanged** (QAbstractSocket::SocketState state)

Private Attributes

QByteArray	incomingBuffer
QByteArray	outgoingBuffer
QList< BlockInfo >	pendingBlocks
int	pendingBlockSizes
QList< TorrentBlock >	incoming
PeerWireState	pwState
bool	receivedHandShake
bool	gotPeerId
bool	sentHandShake
int	nextPacketLength
qint64	uploadSpeedData [8]
qint64	downloadSpeedData [8]
int	transferSpeedTimer
int	timeoutTimer
int	pendingRequestTimer
bool	invalidateTimeout
int	keepAliveTimer
QByteArray	infoHash

QByteArray	peerIdString
QBitArray	peerPieces
TorrentPeer *	torrentPeer
QTcpSocket	socket

הסבר: המחלקה אחראית על ביצועה של תקשורת דו-צדדית בין שני Peers בכדי להעביר חלקי קובץ בין האחד לשני. המחלקה יודעת לנהל events שונים בשיחה בין שני Peers (ולפעול בהתאם) ולנהל את העבודה מול ה-socket (חיבור ל-Peer אחר, קריאה וכתיבה של מידע, ניהול שגיאות וכו').

TorrentClient Class Reference

```
#include <torrentclient.h>
```

Public Types

enum	State {
	Idle, Paused, Stopping, Preparing,
	Searching, Connecting, WarmingUp, Downloading,
	Endgame, Seeding
	}

enum	Error {
	UnknownError, TorrentParseError, InvalidTrackerError, FileError,
	ServerError
	}

Public Slots

void	start ()
void	stop ()
void	setPaused (bool paused)

void	setupIncomingConnection (PeerWireClient *client)

Signals

void	stateChanged (TorrentClient::State state)
void	error (TorrentClient::Error error)
void	downloadCompleted ()
void	peerInfoUpdated ()
void	dataSent (int uploadedBytes)
void	dataReceived (int downloadedBytes)
void	progressUpdated (int percentProgress)
void	downloadRateUpdated (int bytesPerSecond)
void	uploadRateUpdated (int bytesPerSecond)
void	stopped ()

Public Member Functions

	TorrentClient (QObject *parent=0)
	~TorrentClient ()
bool	setTorrent (const QString &fileName)
bool	setTorrent (const QByteArray &torrentData)

MetaInfo	metaInfo () const
void	setMaxConnections (int connections)
int	maxConnections () const
void	setDestinationFolder (const QString &directory)
QString	destinationFolder () const
void	setDumpedState (const QByteArray &dumpedState)
QByteArray	dumpedState () const
qint64	progress () const
void	setDownloadedBytes (qint64 bytes)
qint64	downloadedBytes () const
void	setUploadedBytes (qint64 bytes)
qint64	uploadedBytes () const
int	connectedPeerCount () const
int	seedCount () const
QByteArray	peerId () const
QByteArray	infoHash () const
quint16	serverPort () const
State	state () const

QString	stateString () const
Error	error () const
QString	errorString () const

Protected Slots

void	timerEvent (QTimerEvent *event) Q_DECL_OVERRIDE

Private Slots

void	sendToPeer (int readId, int pieceIndex, int begin, const QByteArray &data)
void	fullVerificationDone ()
void	pieceVerified (int pieceIndex, bool ok)
void	handleFileError ()
void	connectToPeers ()
QList< TorrentPeer * >	weighedFreePeers () const
void	setupOutgoingConnection ()
void	initializeConnection (PeerWireClient *client)
void	removeClient ()
void	peerPiecesAvailable (const QBitArray &pieces)

void	peerRequestsBlock (int pieceIndex, int begin, int length)
void	blockReceived (int pieceIndex, int begin, const QByteArray &data)
void	peerWireBytesWritten (qint64 bytes)
void	peerWireBytesReceived (qint64 bytes)
int	blocksLeftForPiece (const TorrentPiece *piece) const
void	scheduleUploads ()
void	scheduleDownloads ()
void	schedulePieceForClient (PeerWireClient *client)
void	requestMore (PeerWireClient *client)
int	requestBlocks (PeerWireClient *client, TorrentPiece *piece, int maxBlocks)
void	peerChoked ()
void	peerUnchoked ()
void	addToPeerList (const QList< TorrentPeer > &peerList)
void	trackerStopped ()
void	updateProgress (int progress=-1)

Private Attributes

TorrentClientPrivate *	d

Friends

class	TorrentClientPrivate

הסבר: מחלקה זו מהווה את לב המערכת והיא אחראית על ניהול כללי של אובייקטים אחרים (TrackerClient, TorrentServer, PeerWireClient) הקשורים לקיום כלל התקשורת. כמו כן מחלקה זו אחראית על קביעת מצבים ו-events המשפיעים על כלל המערכת.

TorrentClientPrivate Class Reference

Public Member Functions

	TorrentClientPrivate (**TorrentClient** *qq)
void	**setError** (**TorrentClient::Error error**)
void	**setState** (**TorrentClient::State state**)
void	**callScheduler** ()
void	**callPeerConnector** ()

Public Attributes

TorrentClient::Error	**error**
TorrentClient::State	**state**
QString	**errorString**
QString	**stateString**
QString	**destinationFolder**
MetaInfo	**metaInfo**

QByteArray	**peerId**
QByteArray	**infoHash**
TrackerClient	**trackerClient**
FileManager	**fileManager**
QList< **PeerWireClient** * >	**connections**
QList< **TorrentPeer** * >	**peers**
bool	**schedulerCalled**
bool	**connectingToClients**
int	**uploadScheduleTimer**
QMap< int, **PeerWireClient** * >	**readIds**
QMultiMap< **PeerWireClient** *, **TorrentPiece** * >	**payloads**
QMap< int, **TorrentPiece** * >	**pendingPieces**
QBitArray	**completedPieces**
QBitArray	**incompletePieces**
int	**pieceCount**
int	**lastProgressValue**
qint64	**downloadedBytes**
qint64	**uploadedBytes**

int	**downloadRate** [RateControlWindowLength]
int	**uploadRate** [RateControlWindowLength]
int	**transferRateTimer**
TorrentClient *	**q**

הסבר: מחלקה זו הינה מחלקת עזר למחלקה TorrentClient. תפקידה לנהל שגיאות ולקבוע מצבים.

TorrentServer Class Reference

`#include <torrentserver.h>`

Public Member Functions

	TorrentServer ()
void	**addClient** (**TorrentClient** *client)
void	**removeClient** (**TorrentClient** *client)

Static Public Member Functions

static **TorrentServer** *	**instance** ()

Protected Member Functions

void	**incomingConnection** (qintptr socketDescriptor) Q_DECL_OVERRIDE

Private Slots

void	**removeClient** ()

void	processInfoHash (const QByteArray &infoHash)

Private Attributes

QList< TorrentClient * >	clients

הסבר: תפקידה של מחלקה זו הוא לקבל בקשות מ-Peers אחרים כאשר מתבצע Seeding. מחלקה זו מפעילה TorrentClient חדש בכדי לנהל את התקשורת.

MetaInfo Class Reference

```
#include <metainfo.h>
```

Public Types

enum	FileForm { SingleFileForm, MultiFileForm }

Public Member Functions

	MetaInfo ()
void	clear ()
bool	magparse (const QByteArray &data)
QString	errorString () const
QByteArray	infoValue () const
FileForm	fileForm () const
QString	announceUrl () const
QStringList	announceList () const

QDateTime	creationDate () const
QString	comment () const
QString	createdBy () const
MetaInfoSingleFile	singleFile () const
QList< MetaInfoMultiFile >	multiFiles () const
QString	name () const
int	pieceLength () const
QList< QByteArray >	sha1Sums () const
qint64	totalSize () const

Private Attributes

QString	errString
QByteArray	content
QByteArray	infoData
FileForm	metaInfoFileForm
MetaInfoSingleFile	metaInfoSingleFile
QList< MetaInfoMultiFile >	metaInfoMultiFiles
QString	metaInfoAnnounce

QStringList	metaInfoAnnounceList
QDateTime	metaInfoCreationDate
QString	metaInfoComment
QString	metaInfoCreatedBy
QString	metaInfoName
int	metaInfoPieceLength
QList< QByteArray >	metaInfoSha1Sums

הסבר: תפקידה של מחלקה זו הוא לבצע Parsing לבקשות המגיעות מה-Tracker ולשמור את המידע המתקבל.

מודול ניהול קבצים:

FileManager Class Reference

`#include <filemanager.h>`

Classes

struct	ReadRequest
struct	WriteRequest

Public Slots

void	startDataVerification ()

Signals

void	**dataRead** (int id, int pieceIndex, int offset, const QByteArray &data)
void	**error** ()
void	**verificationProgress** (int percent)
void	**verificationDone** ()
void	**pieceVerified** (int pieceIndex, bool verified)

Public Member Functions

	FileManager (QObject *parent=0)
virtual	**~FileManager** ()
void	**setMetaInfo** (const **MetaInfo** &info)
void	**setDestinationFolder** (const QString &directory)
int	**read** (int pieceIndex, int offset, int length)
void	**write** (int pieceIndex, int offset, const QByteArray &data)
void	**verifyPiece** (int pieceIndex)
qint64	**totalSize** () const
int	**pieceCount** () const
int	**pieceLengthAt** (int pieceIndex) const
QBitArray	**completedPieces** () const
void	**setCompletedPieces** (const QBitArray &pieces)

QString	**errorString** () const

Protected Member Functions

void	**run** () Q_DECL_OVERRIDE

Private Slots

bool	**verifySinglePiece** (int pieceIndex)
void	**wakeUp** ()

Private Member Functions

bool	**generateFiles** ()
QByteArray	**readBlock** (int pieceIndex, int offset, int length)
bool	**writeBlock** (int pieceIndex, int offset, const QByteArray &data)
void	**verifyFileContents** ()

Private Attributes

QString	**errString**
QString	**destinationPath**
MetaInfo	**metaInfo**
QList< QFile * >	**files**

QList< QByteArray >	sha1s
QBitArray	verifiedPieces
bool	newFile
int	pieceLength
qint64	totalLength
int	numPieces
int	readId
bool	startVerification
bool	quit
bool	wokeUp
QList< WriteRequest >	writeRequests
QList< ReadRequest >	readRequests
QList< int >	pendingVerificationRequests
QList< int >	newPendingVerificationRequests
QList< qint64 >	fileSizes
QMutex	mutex
QWaitCondition	cond

הסבר: מחלקה זו אחראית על התנהלות מול קבצים. באמצעות מחלקה זו ניתן לבדוק את תקינות הקובץ, לקרוא מקובץ ולכתוב אל תוך קובץ, בחלקים וכן תוך כדי תמיכה בקבצים גדולים.

מודול הגדרות:

Settings Class Reference

`#include <settings.h>`

Static Public Member Functions

static void	load ()
static void	save ()
static void	assign (QString host, int aPort, QString aFile, QString aLevel, QString aFolder, QString aMSF)
static int	trackerPort ()
static QString	trackerHost ()
static QString	logFile ()
static QString	logLevel ()
static QString	destinationFolder ()
static QString	msfFolder ()

Static Public Attributes

static const QString	INI_FILE = "magshare.conf"

Static Private Attributes	
static int	theTrackerPort
static QString	theTrackerHost
static QString	theLogFile
static QString	theLogLevel
static QString	theDestinationFolder
static QString	theMSFFolder

הסבר: המחלקה אחראית על טעינה ושמירה של הגדרות בקובץ magshare.conf ומתן המידע לאובייקטים שונים במערכת הזקוקים להגדרות אלו.

מודול UI:

MainWindow Class Reference

```
#include <mainwindow.h>
```

Classes	
struct	Job

Public Member Functions	
	MainWindow (QWidget *parent=0)
	~MainWindow ()

void	**torrentStart** ()
void	**torrentStop** ()

Private Slots

void	**on_actionAbout_triggered** ()
void	**on_actionExit_triggered** ()
void	**on_actionSettings_triggered** ()
void	**on_actionAdd_MagFile_triggered** ()
void	**on_actionCreate_MagFile_triggered** ()
void	**on_pushStart_clicked** ()
void	**on_pushStop_clicked** ()
bool	**addTorrent** (const QString &fileName, const QString &destinationFolder, const QByteArray &resumeState=QByteArray())
void	**removeTorrent** ()
void	**torrentError** (**TorrentClient::Error** error)
void	**torrentStopped** ()
void	**updateState** (**TorrentClient::State** state)
void	**updatePeerInfo** ()
void	**updateProgress** (int percent)

void	updateDownloadRate (int bytesPerSecond)
void	updateUploadRate (int bytesPerSecond)

Private Member Functions

int	rowOfClient (TorrentClient *client) const

Private Attributes

Ui::MainWindow *	ui
int	uploadLimit
int	downloadLimit
QList< Job >	jobs
int	jobsStopped
int	jobsToStop
QString	lastDirectory

הסבר: המחלקה אחראית על ממשק המשתמש של החלון המרכזי של הקליינט, כולל תפריט לניהול מסכים נוספים והפעלה ועצירה של יישומי המערכת.

SettingsDialog Class Reference

```
#include <settingsdialog.h>
```

Public Member Functions

	SettingsDialog (QWidget *parent=0)

	~SettingsDialog ()

Private Slots

void	on_buttonBox_accepted ()
void	on_pushDestFolder_clicked ()
void	on_pushMSFfolder_clicked ()

Private Member Functions

void	assignSettings ()
void	keepSettings ()

Private Attributes

Ui::Settings *	ui

הסבר: המחלקה אחראית על הצגת הגדרות המערכת ומאפשרת את שינויים ועדכונם דרך המחלקה Settings.

AboutDialog Class Reference

```
#include <aboutdialog.h>
```

Public Member Functions

	AboutDialog (QWidget *parent=0)
	~AboutDialog ()

Private Attributes	
Ui::AboutDialog *	ui

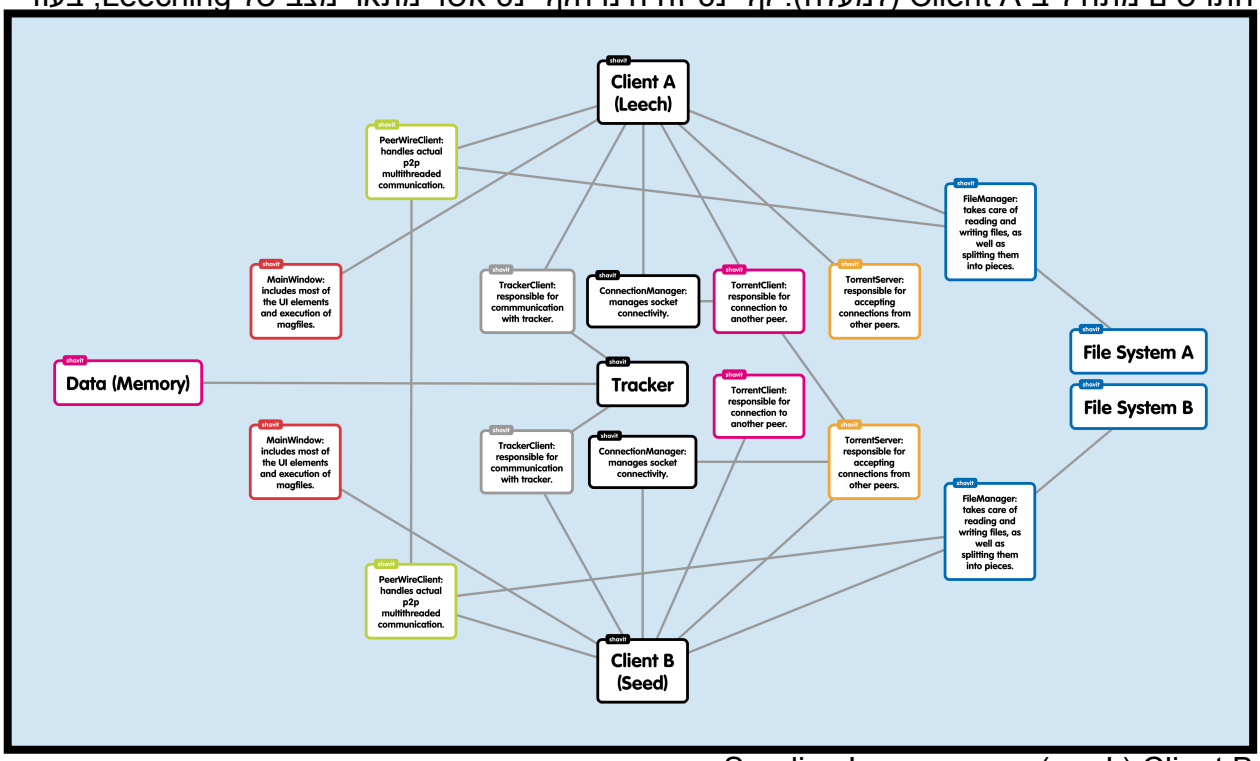
הסבר: המחלקה אחראית על הצגת חלון ה-About.

Logging – משותף ל-Client ו-Tracker

במערכת קיימות מספר מחלקות האחראיות על logging. מחלקות אלה נלקחו מפרויקט בשם QsLog אשר מצוי ב-Github (https://github.com/victronenergy/QsLog). הסבר על פרויקט זה מצורף בנספחים.

2.3. דיון לגבי העיצוב הנבחר

התרשים הנ"ל (נמצא גם בנספחים) מתאר את האופן בו עובדת המערכת:
התרשים מתחיל ב-Client A (למעלה). קליינט זה הינו הקליינט אשר מתאר מצב של Leeching, בעוד



Client B (למטה) מתאר מצב של Seeding.

לכל קליינט קיימת האפשרות ליצור קבצים, באמצעות FileManager (בכחול מימין). מחלקה זו מדברת אל מערכת הקבצים המקומית בכדי ליצור, לקרוא ולכתוב לקבצים. לכל קליינט קיים UI (MainWindow, באדום משמאל).

Leeching מתאפשר כאשר ה-Client מזהה שהקובץ המיועד (אליו מפנה קובץ ה-MSF שנפתח) איננו קיים בתקיית הקבצים המיועדת. כאשר אנו רוצים לבצע Leeching, נפנה, לאחר פתיחת קובץ ה-MSF, ל-TrackerClient. מחלקה זו יודעת לתקשר אל מול ה-Tracker. בעזרת מחלקה זו נשלח בקשה ל-Tracker שימצא עבורנו Peer מתאים להורדת הקובץ. במקביל לשליחת הבקשה, FileManager מייצר קובץ ריק בתקייה המיועדת אשר ממולא ב-NULLים בגודל הקובץ אותו ברצוננו להוריד.

הבקשה ל-Tracker נשלחת באמצעות Socket וכוללת בתוכה שני חלקים עיקריים; חלק בעל מידע על הקובץ במשותף (info) וחלק בעל זיהוי ייחודי חד-דרכי של הקובץ (דומה ל-checksum) שתוכנו הוא מחרוזת Hash (*ראה בהמשך:* "3. עיצוב הנתונים (data)"). בנוסף, הבקשה כוללת את כמות הבתים של הקובץ שכבר מצויים במערכת הקבצים וכתובת IP, כמו גם port, של ה-Client. בשל השימוש ב-Socketים, שינוי ה-IP של משתמשים שונים לא נעשה תוך כדי התחברות ל-Tracker, משום שה-Session הינו יחיד.

ל-Tracker קיימת בזמן אמת רשימה בשם MetaInfoStruct, השמורה בזיכרון (ורוד משמאל), של Peers פעילים. הרשימה כוללת את השדות הבאים:

- `name` – שם הקובץ.
- `length` – גודל הקובץ בבתים.
- `pieceLength` – גודלו של כל חלק של הקובץ.
- `sha1Sums` – Hash של כלל חלקי הקובץ.
- `peerid` – מזהה ייחודי של ה-Peer הנוכחי.
- `port` – פורט של החיבור.
- `ip` – כתובת IP של ה-Peer.
- `trackerid` – מזהה שה-Tracker נותן לכל Peer שמתחבר אליו.
- `lastUpd` – Timestamp של חיבורו האחרון של ה-Peer ל-Tracker.

ה-Tracker ישווה את ה-Hash של קובץ ה-MSF שנשלח מהקליינט עם ה-Hash של כל Peer בכדי למצוא Client מתאים לבקשה. במידה ונמצאו Peers מתאימים, תוחזר רשימת Peers מלאה ל-Client (שידע לאחר מכן לבחור מאיזה מהם להתחבר).

רשימה זו של Peers כוללת את השדות הבאים שנלקחו מתוך ה-MetalInfoStruct שמחזיק ה-Tracker עבור כל Peer: peerid, ip ו-port.

כאשר לקליינט יש Peer פוטנציאלי להתחברות, הוא ישתמש במחלקה TorrentClient (ורוד באמצע) בכדי לשלוח בקשה ל-Peer הנבחר, אשר יקבל את הבקשה דרך המחלקה TorrentServer (כתום באמצע), שתפקידה להאזין לבקשות נכנסות מצד Peers אחרים. חיבור זה ינוהל באמצעות ConnectionManager (שחור באמצע).

במידה ונוצר חיבור תקין, יחלו שני ה-Peers בהעברת המידע ביניהם, באמצעות המחלקה PeerWireClient (ירוק בשמאל) בכל אחד מהם. בצד שמבצע Seeding, תקרא כל פעם מחלקה זו חלק מהקובץ שנמצא במערכת הקבצים של המחשב עליו היא מופעלת ותעביר אותו לקליינט השני. הצד

שמבצע Leeching יכתוב בכל פעם את החלק אותו הוא יקבל אל הקובץ (אשר בתחילה ממולא ב-NULLים), גם באמצעות מחלקה זו.

במשך כל התהליך התקשורת מתוחזקת ומתבצעת בדיקה מתמדת שהיא אכן עובדת (במידה ולא, ידע הקליינט לבקש בקשות מחודשות ל-Tracker בכדי לקבל Peers אחרים אשר עובדים).

Seeding מתקיים כאשר נפתח ב-Client קובץ MSF וה-Client מזהה את הקובץ המיועד בתקייה המוגדרת. במקרה זה, ה-Tracker יידע שמדובר ב-Peer שמבצע Seeding, וה-Client יעבור למצב של האזנה לבקשות מצד Peers אחרים.

3. עיצוב הנתונים (data)

במערכת MagShare נשמרים הנתונים עבור קבצים המיועדים לשיתוף בקובץ מיוחד בשם קובץ MSF.
עיצוב הנתונים של קובץ MSF הוא כדלהלן:

קובץ MSF בנוי ממספר חלקים. החלקים מופרדים ביניהם בתו '&' (אמפרסנד). הפורמט עבור כל חלק הוא:

```
part-name=value
```

להלן דוגמא – ניקח את קובץ ההרצה של המשחק Minecraft (הקובץ הוא MinecraftLauncher.exe) וניצור ממנו קובץ MSF, לו נקרא minecraft.msf. תוכנו של הקובץ ייראה כך:

```
mfile-name=MinecraftLauncher.exe&mfile-length=1247112&mfile-piece-length=1048576&mfile-num-pieces=2&mfile-pieces=AoPldut4jhKHNeA9csWowtDoSi8%3D5x0BB1CkDvyUaVeDbYPBpQw9Y0c%3D
```

חלקיו השונים של הקובץ סומנו בצבעים שונים. החלקים הם:

- **mfile-name**: שם הקובץ המקורי.
- **mfile-length**: גודלו של הקובץ בבתים.
- **mfile-piece-length**: גודלו של כל חלק של הקובץ.
- **mfile-num-pieces**: מספר החלקים אליהם חולק הקובץ.
- **mfile-pieces**: מחרוזת Hash מסוג SHA1 של הבתים המרכיבים את הקובץ, לאחר Encoding ב-Base64.

עיצוב אחיד זה של הנתונים מאפשר השוואה בין הקבצים, כאשר ה-Tracker מחפש Peer עם הקובץ הרצוי להורדה. כמו כן, ה-Encoding מקל על העברת הנתונים משום שקל יותר להעביר תווים של UTF-8 מאשר רצף בינארי.

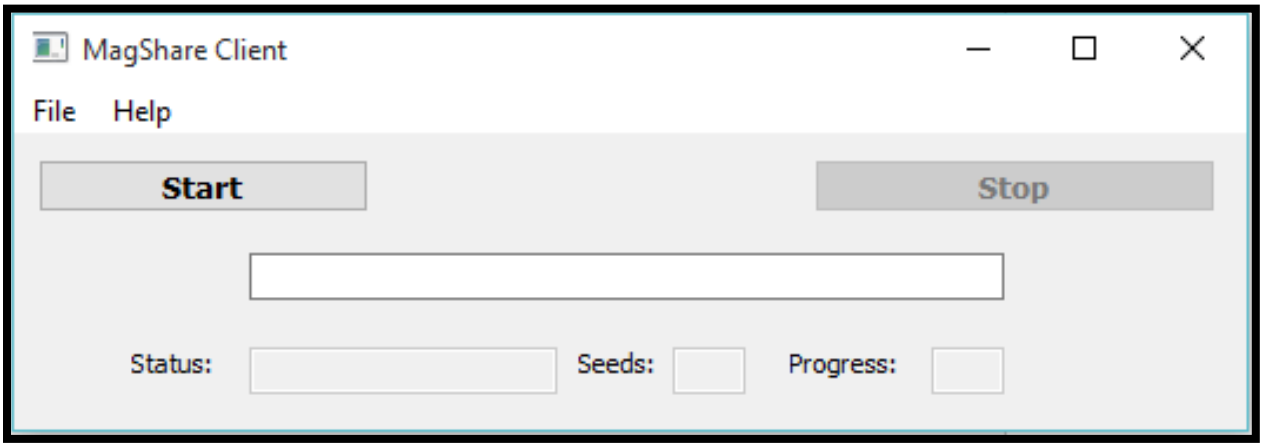
קובץ נוסף אשר נשמר במערכת הוא קובץ ההגדרות. קובץ זה מכיל בתוכו את ההגדרות שנקבעו בחלון Settings (מפורט בהמשך). להלן דוגמא לתוכנו של קובץ הגדרות:

```
[Tracker]
Host=localhost
Port=1337

[Local]
LogFile=magshare.log
LogLevel=TRACE
DestinationFolder=C:/Magshare/src
MSFFolder=C:/Magshare/msf
```

בנוסף, נשמר קובץ log עבור הקליינט וקובץ log נוסף עבור ה-Tracker. קבצים אלה מכילים את כלל הודעות המערכת שהוצאו בהתאם לרמת ה-logging שנקבעה החלון ה-Settings (מפורט בהמשך).

4. ממשק משתמש

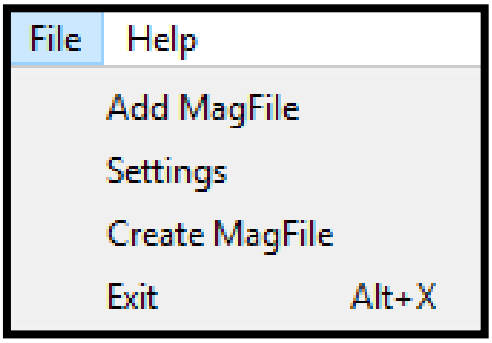


זהו המסך הראשי של התכנה, אותו רואה כל משתמש כאשר הוא מפעיל אותה. מסך זה הוא הקליינט (ל-Tracker אין ממשק משתמש). במסך זה ניתן להתחיל Leeching או Seeding של קובץ, על ידי לחיצה על כפתור Start לאחר בחירה של קובץ MSF. ניתן גם לעצור תהליך זה על ידי לחיצה על כפתור Stop. אם המערכת מזהה שהקובץ אליו מפנה ה-MSF כבר נמצא במערכת הקבצים (בתקיה שהוגדרה בדף הגדרות, מפורט בהמשך), מתבצע Seeding; אחרת – מתבצע Leeching.

לאחר תחילת התהליך, מתעדכן שדה ה-Status מול ה-Tracker (אפשרויות שונות ל-Status הן Searching, Connecting, Downloading, Uploading ועוד).

השדה Seeds רלוונטי רק כאשר מתבצע Leeching והוא מציג את מספר ה-Peers המחזיקים בקובץ.

השדה Progress רלוונטי גם הוא רק כאשר מתבצע Leeching והוא מציג מספר בין 0 ל-100 המייצג את אחוז ההורדה אשר כבר התבצע.

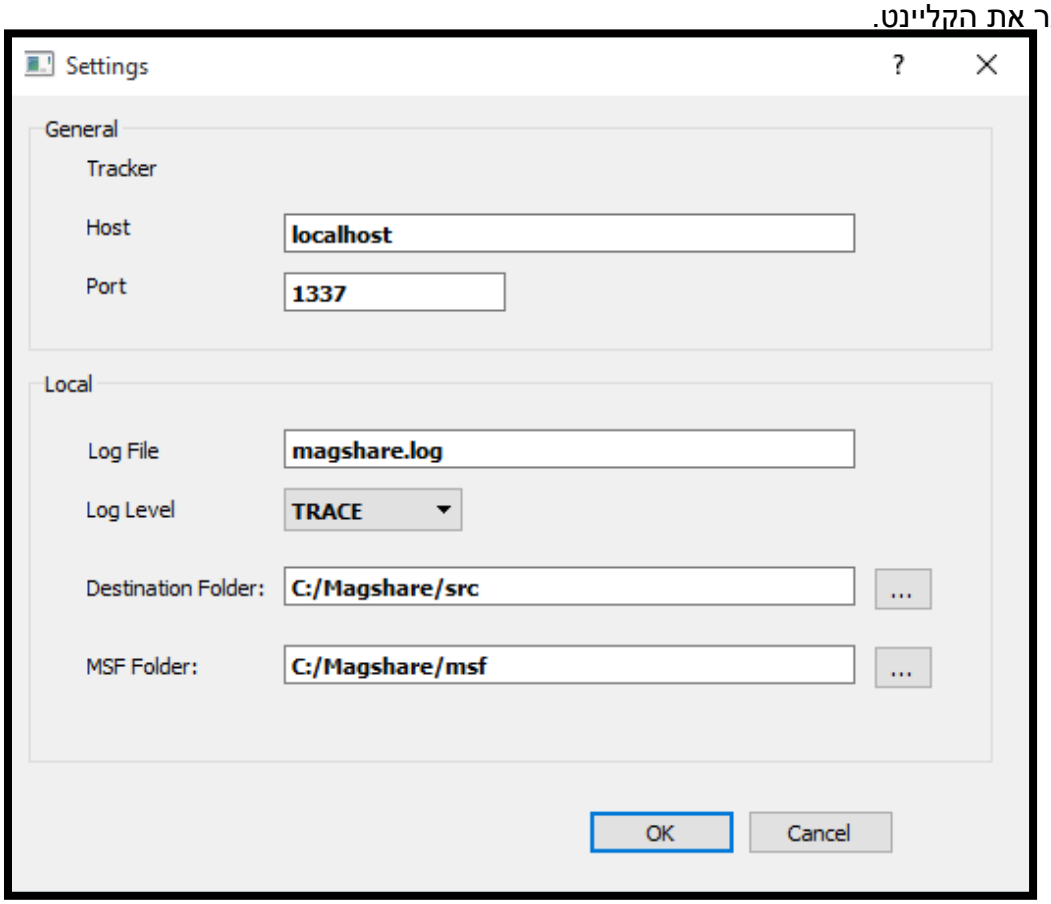


זהו המסך המתקבל כאשר לוחצים על File (מצד שמאל). Add MagFile יפתח חלון בו ניתן לבחור קבצים מסוג MSF, בכדי להתחיל Leeching או Seeding.

Settings יפתח חלון ובו הגדרות התכנה (מפורט בהמשך).

Create MagFile יפתח חלון ובו ניתן לבחור קובץ מכל סוג שהוא; לאחר בחירת הקובץ יפתח חלון נוסף בו ניתן יהיה לשמור את קובץ ה-MSF שנוצר מהקובץ הנבחר.

Exit סוגר את הקליינט.



זהו החלון שמופיע כאשר לוחצים על Settings מתוך התפריט File. בחלון זה ניתן להזין את הגדרות ה-Tracker (Host ו-Port), להגדיר את קובץ ה-log ואת רמת ה-logging (INFO, TRACE, DEBUG, WARN, ERROR, FATAL).

בנוסף ניתן לקבוע את תקיית היעד של הקבצים המיועדים ל-Seeding או היכן ישמרו הקבצים אותם מורידים וכן לקבוע את התקייה בה יישמרו קבצי ה-MSF.

בתפריט Help קיים מסך About המציג מידע על התכנה, מפתחיה וגרסתה הנוכחית.

5. נספחים

- ממשק html המכיל בתוכו את כלל הקוד כולל חלוקה למחלקות מצורף בתקיה Reference.
- AdvancedLogic.png – דיאגרמת Data Flow – מצורף.
- QsLog.txt – הסבר על פרויקט QsLog.