“Help, Hiking made easy”

Prototype 4.0 Logo

H.E.L.P

Hiking Made Easy

Group G: Andy, Aria, Michael, Malik

Publishing the Application

Once we finish our program and run our final tests, we will start processing the publication aspect of the project. We plant to publish the application onto the Google Play store since it's less rigorous than the Apple store.

Steps:

1. Select All apps > Create app.
2. Select a default language and add a title for your app. Type the name of your app as you want it to appear on Google Play. You can change this later.
3. Specify whether your application is an app or a game. You can change this later.
4. Specify whether your application is free or paid.
5. Add an email address that Play Store users can use to contact you about this application.
6. Complete the “Content guidelines” and “US export laws” declarations.
7. Select Create app

Monetizing our Application

Once our app is created and published in the Google store, we plan to monetize our program.

-We will provide our application as a free download so everyone can use it. The more downloads we have the better.

- Once users create a free account they will be able to choose a standard account which will have paid advertisement which the user must watch, or a premier account for $5.99 which is add free.

-We are hoping to receive paid sponsors once they see our large database of users. Preferably sponsors who are related to hiking such as fitness clothing, energy bars, and etc. however we are open to all sponsors.

OR

-Pay for Tracker at a certain fee

What Has Been Done (Part 1) :

- Login Screen
  - Allows user to login using a username and password
- Create Account
  - Allows user to create an account using first name,last name, email, etc..
- Display All Hikes
  - Displays hikes taken from database
- Forgot Password
  - Allows user to reset password
- Update Account
  - Allows user to update their account of saved hikes
- Delete Account
  - Allows user to delete their account
- Expand View
  - Allows user to see details on specific hikes
- Saving Hikes
  - Allows user to save choose and save their hikes

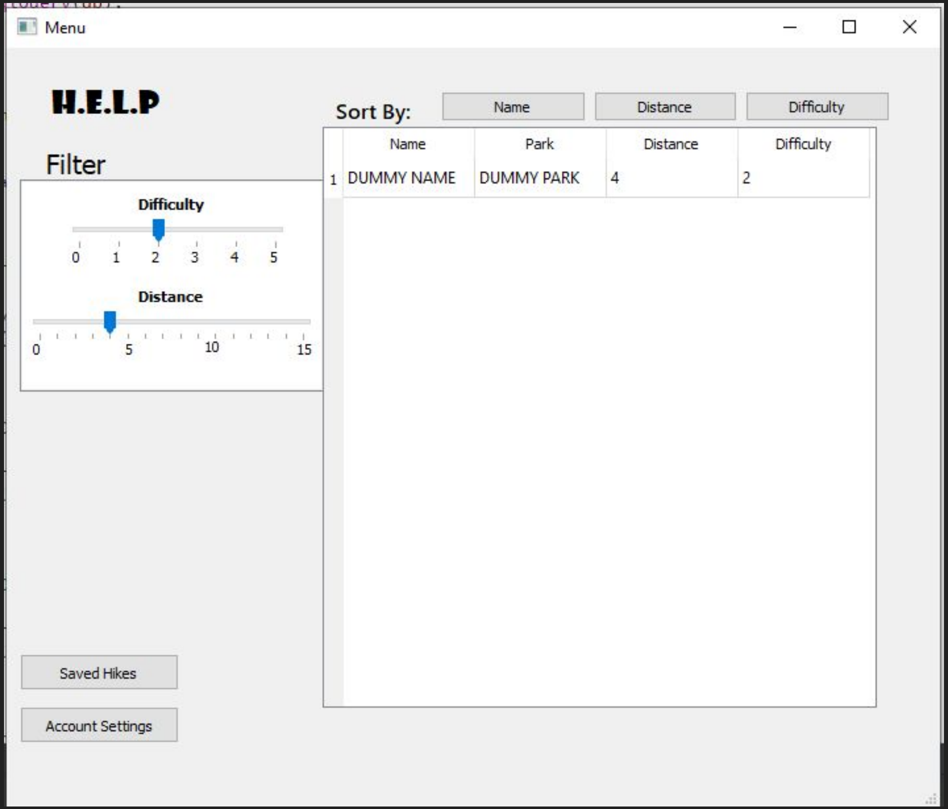
What Has Been Done (Part 2) :

- Admin Window
  - EDIT hikes
    - Displays chosen hike to edit and will be updated database
  - DELETE hikes
    - Displays chosen hike to delete and will be updated database
  - ADD hikes
    - Displays chosen hike to add and will be updated database
  - SEARCH hikes
    - Can search anything pertaining to the hikes

Iteration Cycle 4 Progress

- Filter Queries
  - Allows user to view a segmented list based on qualifiers given.
- Admin Window
  - Change User Interface
  - ADD and EDIT hikes - add and change pictures in the database
  - Can add and/or change pictures from computer via directory path of device
- TRACKER
  - Track time/distance traveled/ speed
  - Tracks user’s coordinates (longitude and latitude)
- Implemented Google Maps
  - Mostly functional widget available when the menu is accessed.

Search Filter



Menu

H.E.L.P

Filter

Difficulty

0 1 2 3 4 5

Distance

0 5 10 15

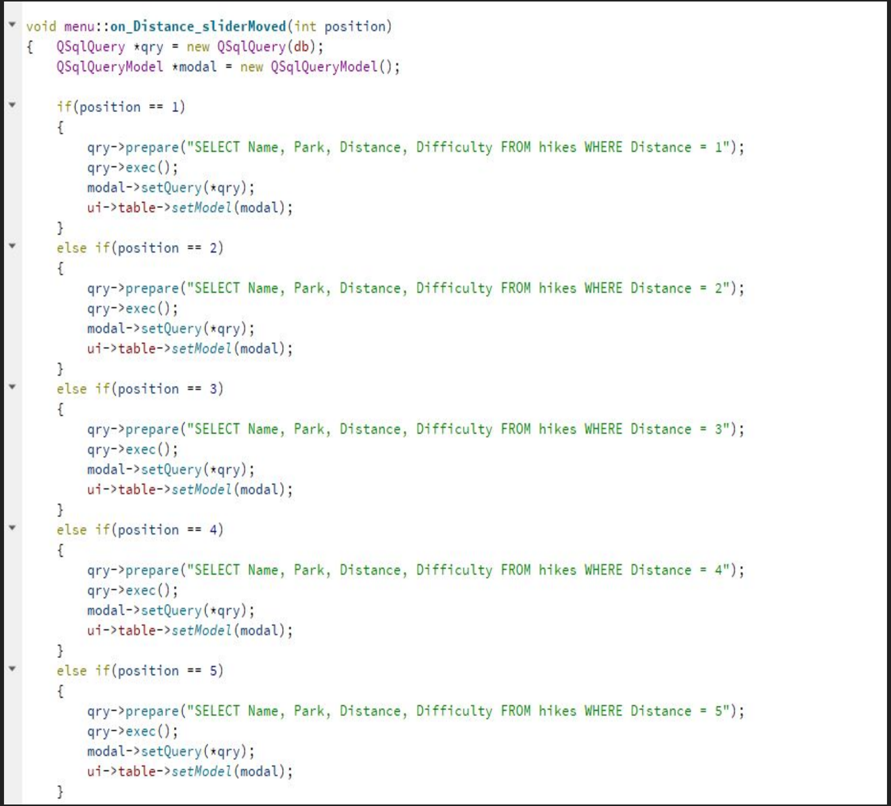
Sort By: Name Distance Difficulty

	Name	Park	Distance	Difficulty
1	DUMMY NAME	DUMMY PARK	4	2

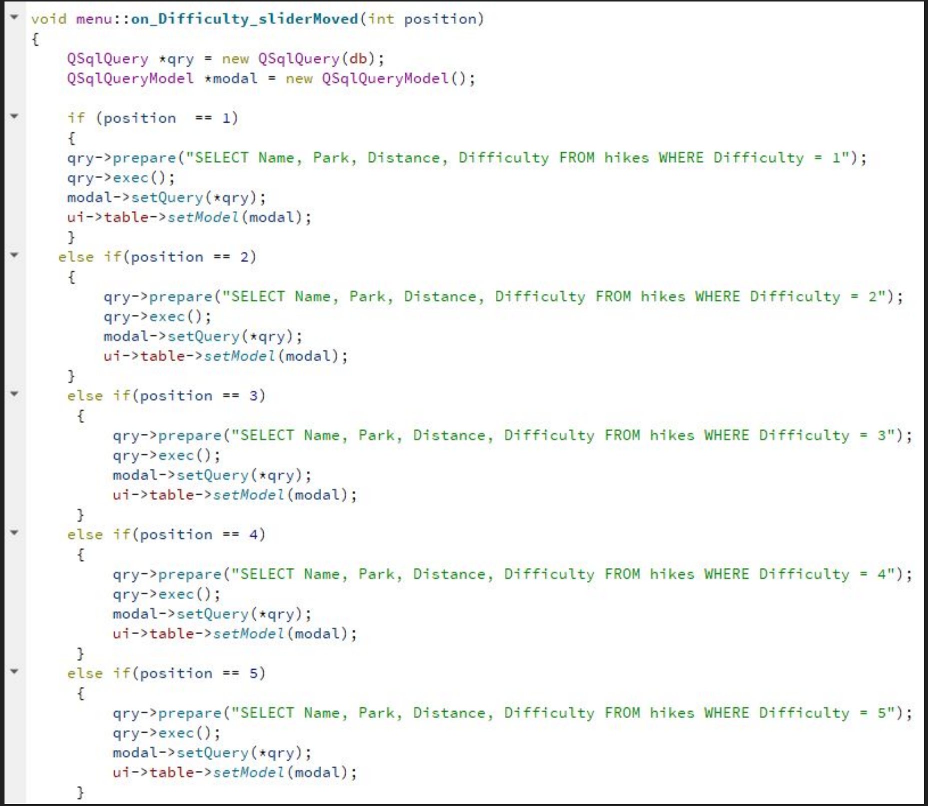
Saved Hikes

Account Settings

Search Filter

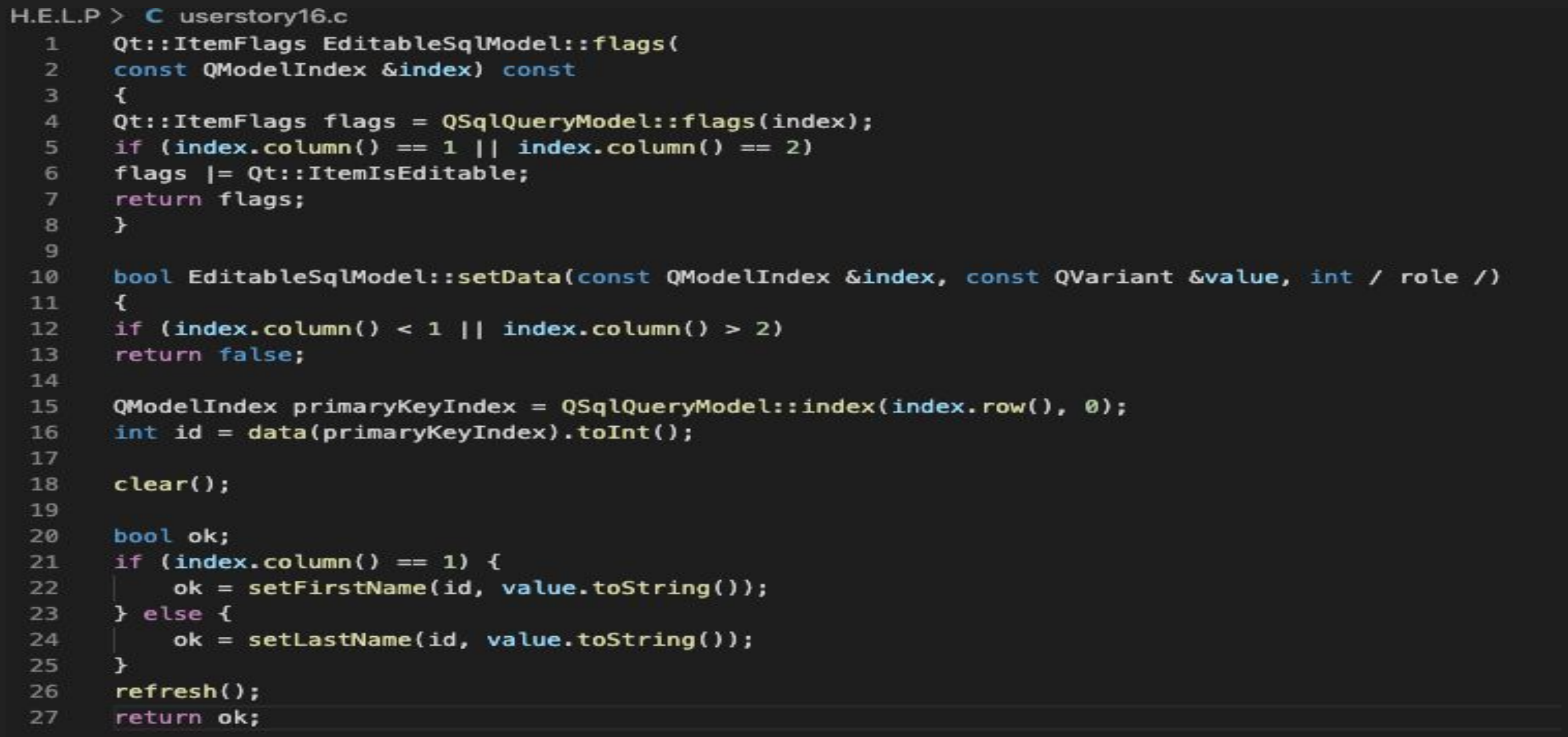


```
void menu::on_Distance_sliderMoved(int position)
{   QSqlQuery *qry = new QSqlQuery(db);
    QSqlQueryModel *modal = new QSqlQueryModel();

    if(position == 1)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Distance = 1");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
    else if(position == 2)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Distance = 2");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
    else if(position == 3)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Distance = 3");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
    else if(position == 4)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Distance = 4");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
    else if(position == 5)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Distance = 5");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
```



```
void menu::on_Difficulty_sliderMoved(int position)
{
    QSqlQuery *qry = new QSqlQuery(db);
    QSqlQueryModel *modal = new QSqlQueryModel();

    if (position  == 1)
    {
    qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Difficulty = 1");
    qry->exec();
    modal->setQuery(*qry);
    ui->table->setModel(modal);
    }
   else if(position == 2)
    {
        qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Difficulty = 2");
        qry->exec();
        modal->setQuery(*qry);
        ui->table->setModel(modal);
    }
    else if(position == 3)
     {
         qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Difficulty = 3");
         qry->exec();
         modal->setQuery(*qry);
         ui->table->setModel(modal);
     }
    else if(position == 4)
     {
         qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Difficulty = 4");
         qry->exec();
         modal->setQuery(*qry);
         ui->table->setModel(modal);
     }
    else if(position == 5)
     {
         qry->prepare("SELECT Name, Park, Distance, Difficulty FROM hikes WHERE Difficulty = 5");
         qry->exec();
         modal->setQuery(*qry);
         ui->table->setModel(modal);
     }
```

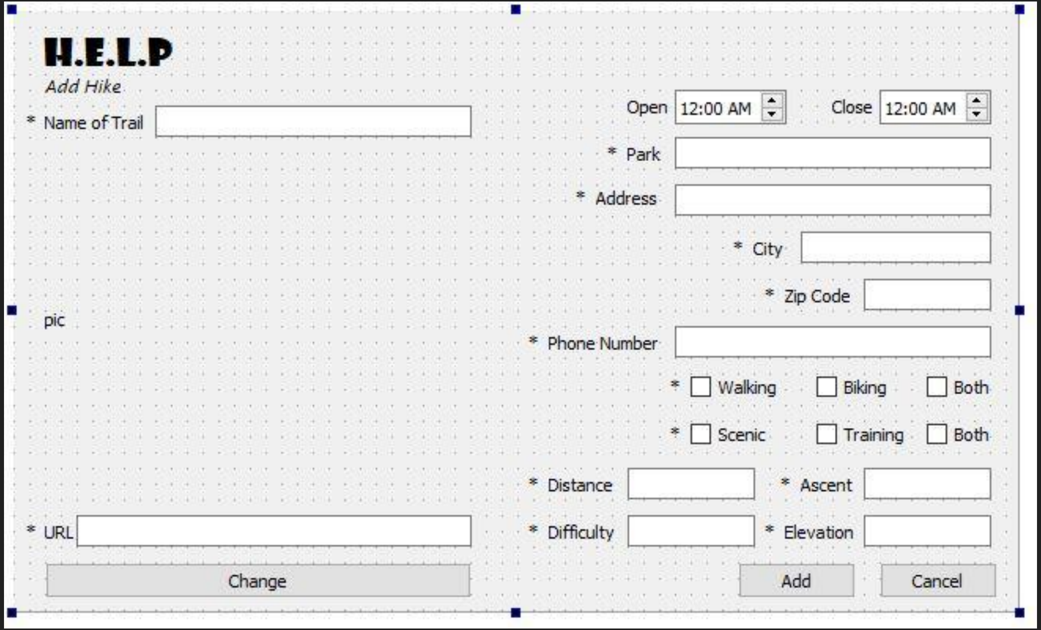
Hike in Detail

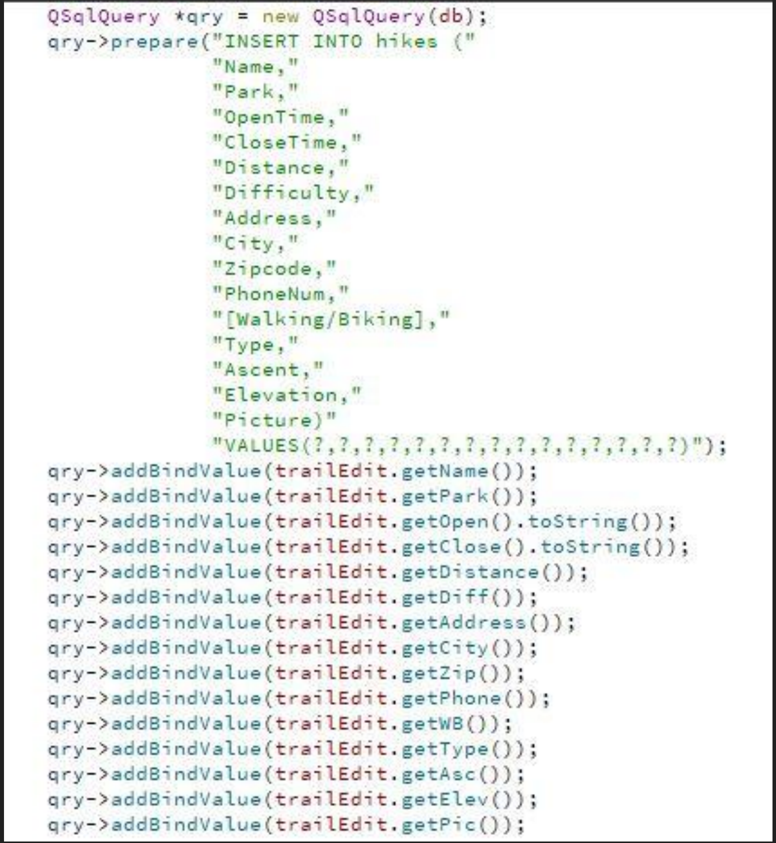


H.E.L.P > C userstory16.c

```
Qt::ItemFlags EditableSqlModel::flags(
const QModelIndex &index) const
{
Qt::ItemFlags flags = QSqlQueryModel::flags(index);
if (index.column() == 1 || index.column() == 2)
flags |= Qt::ItemIsEditable;
return flags;
}

bool EditableSqlModel::setData(const QModelIndex &index, const QVariant &value, int / role /)
{
if (index.column() < 1 || index.column() > 2)
return false;

QModelIndex primaryKeyIndex = QSqlQueryModel::index(index.row(), 0);
int id = data(primaryKeyIndex).toInt();

clear();

bool ok;
if (index.column() == 1) {
    ok = setFirstName(id, value.toString());
} else {
    ok = setLastName(id, value.toString());
}
refresh();
return ok;
```

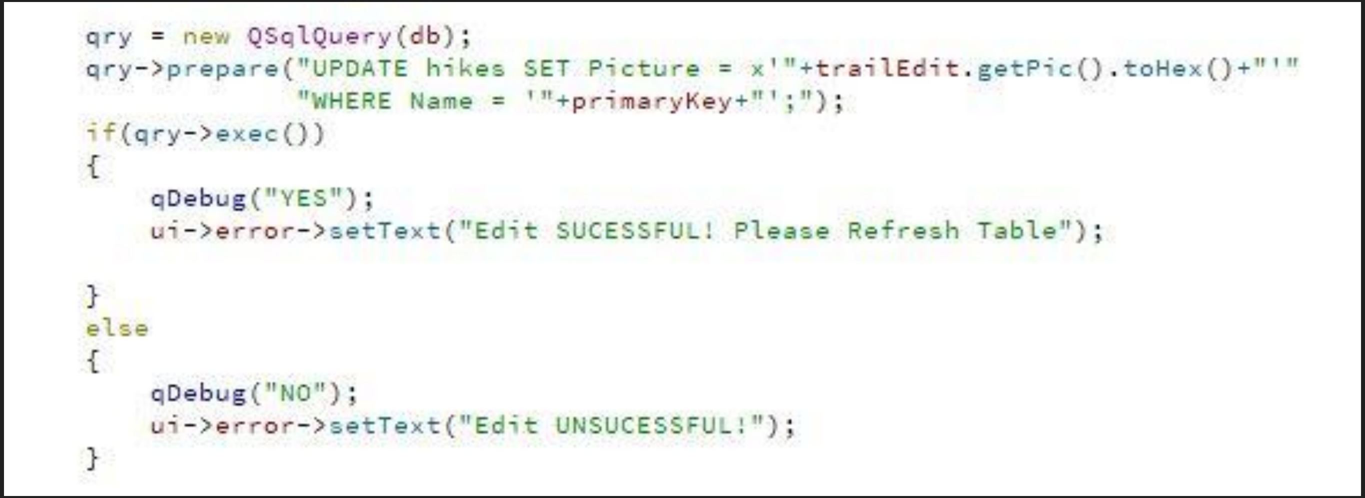
ADD Hikes





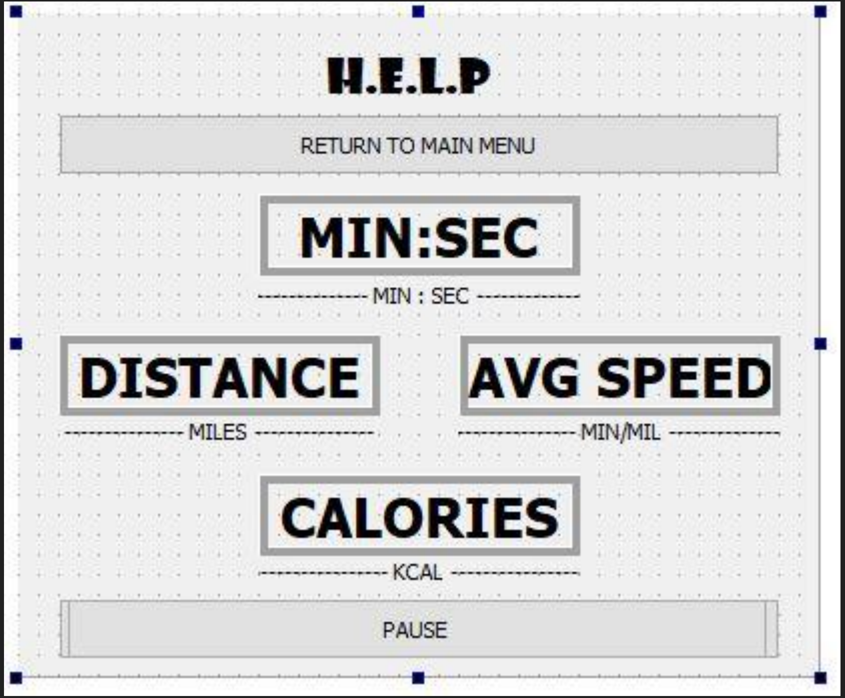
```
QSqlQuery *qry = new QSqlQuery(db);
qry->prepare("INSERT INTO hikes ("
             "Name,"
             "Park,"
             "OpenTime,"
             "CloseTime,"
             "Distance,"
             "Difficulty,"
             "Address,"
             "City,"
             "Zipcode,"
             "PhoneNum,"
             "[Walking/Biking],"
             "Type,"
             "Ascent,"
             "Elevation,"
             "Picture)"
             "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
qry->addBindValue(trailEdit.getName());
qry->addBindValue(trailEdit.getPark());
qry->addBindValue(trailEdit.getOpen().toString());
qry->addBindValue(trailEdit.getClose().toString());
qry->addBindValue(trailEdit.getDistance());
qry->addBindValue(trailEdit.getDiff());
qry->addBindValue(trailEdit.getAddress());
qry->addBindValue(trailEdit.getCity());
qry->addBindValue(trailEdit.getZip());
qry->addBindValue(trailEdit.getPhone());
qry->addBindValue(trailEdit.getWB());
qry->addBindValue(trailEdit.getType());
qry->addBindValue(trailEdit.getAsc());
qry->addBindValue(trailEdit.getElev());
qry->addBindValue(trailEdit.getPic());
```

EDIT Hikes

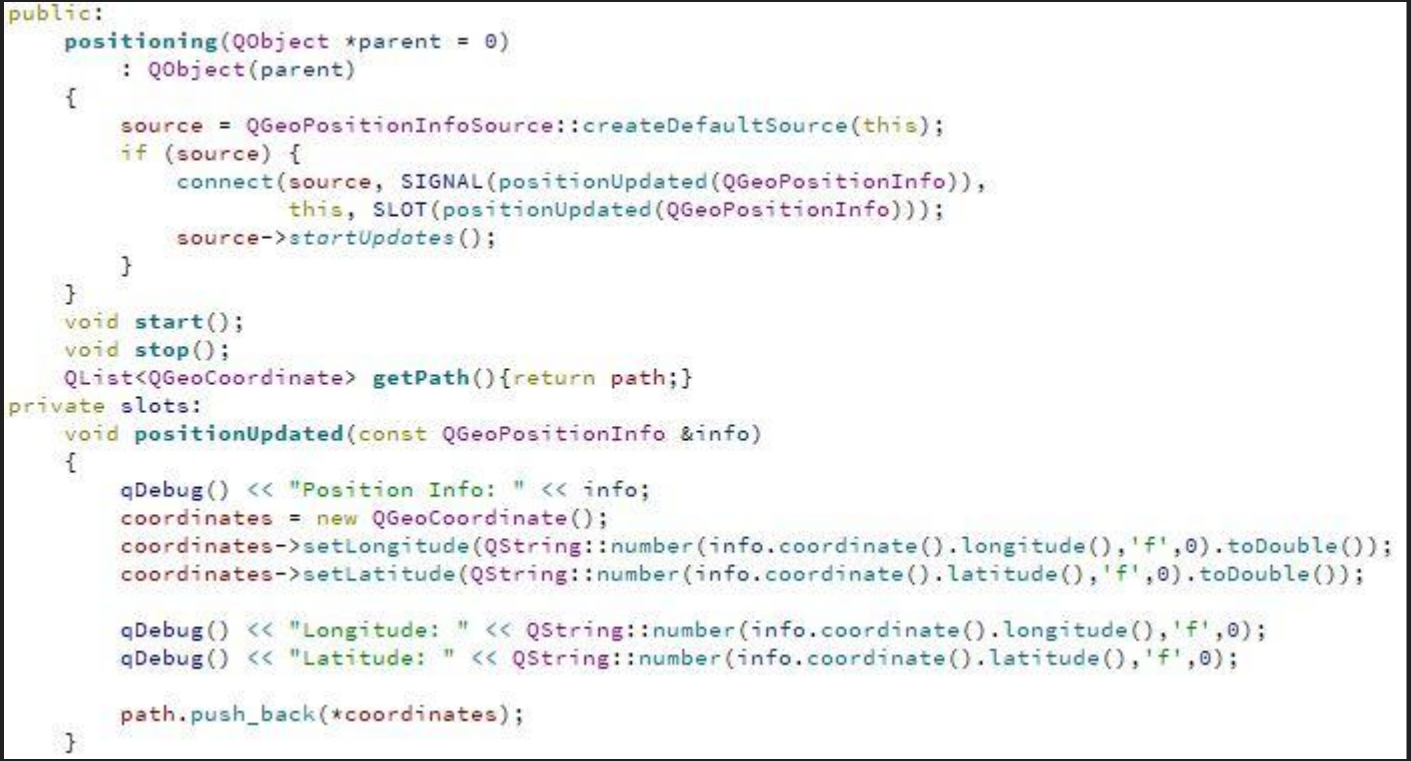


```
qry = new QSqlQuery(db);
qry->prepare("UPDATE hikes SET Picture = x'"+trailEdit.getPic().toHex()+"'"
             "WHERE Name = '"+primaryKey+"';");
if(qry->exec())
{
    qDebug("YES");
    ui->error->setText("Edit SUCESSFUL! Please Refresh Table");

}
else
{
    qDebug("NO");
    ui->error->setText("Edit UNSUCESSFUL!");
}
```

Tracker

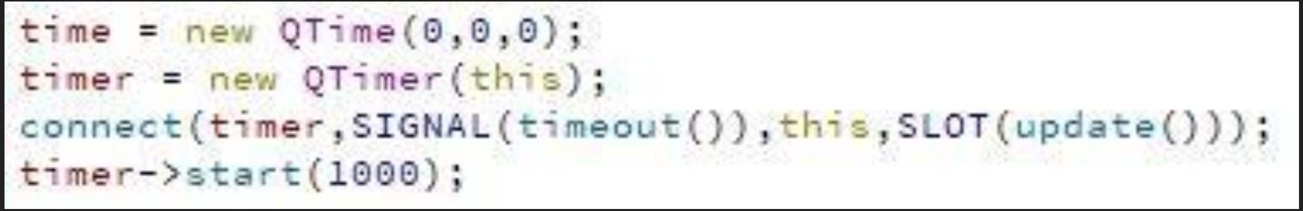

H.E.L.P
RETURN TO MAIN MENU
MIN:SEC
MIN : SEC
DISTANCE
MILES
AVG SPEED
MIN/MIL
CALORIES
KCAL
PAUSE

User’s Coordinates

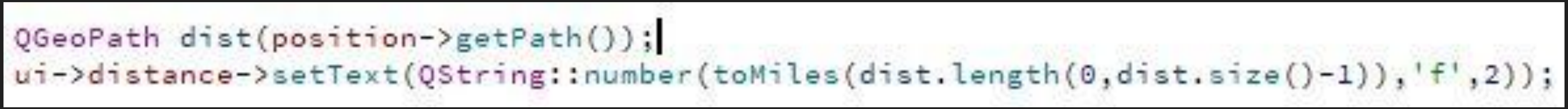


```
public:
    positioning(QObject *parent = 0)
        : QObject(parent)
    {
        source = QGeoPositionInfoSource::createDefaultSource(this);
        if (source) {
            connect(source, SIGNAL(positionUpdated(QGeoPositionInfo)),
                    this, SLOT(positionUpdated(QGeoPositionInfo)));
            source->startUpdates();
        }
    }
    void start();
    void stop();
    QList<QGeoCoordinate> getPath(){return path;}
private slots:
    void positionUpdated(const QGeoPositionInfo &info)
    {
        qDebug() << "Position Info: " << info;
        coordinates = new QGeoCoordinate();
        coordinates->setLongitude(QString::number(info.coordinate().longitude(),'f',0).toDouble());
        coordinates->setLatitude(QString::number(info.coordinate().latitude(),'f',0).toDouble());

        qDebug() << "Longitude: " << QString::number(info.coordinate().longitude(),'f',0);
        qDebug() << "Latitude: " << QString::number(info.coordinate().latitude(),'f',0);

        path.push_back(*coordinates);
    }
```

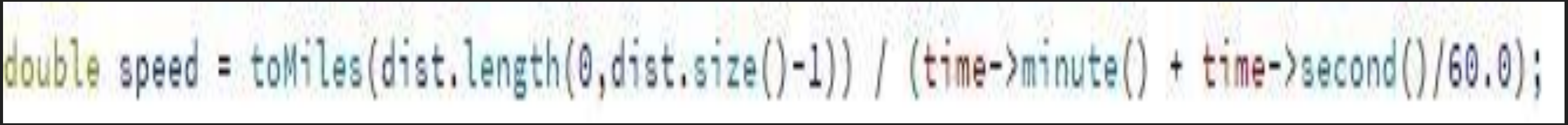
Time/Distance/Speed



```
time = new QTime(0,0,0);
timer = new QTimer(this);
connect(timer,SIGNAL(timeout()),this,SLOT(update()));
timer->start(1000);
```



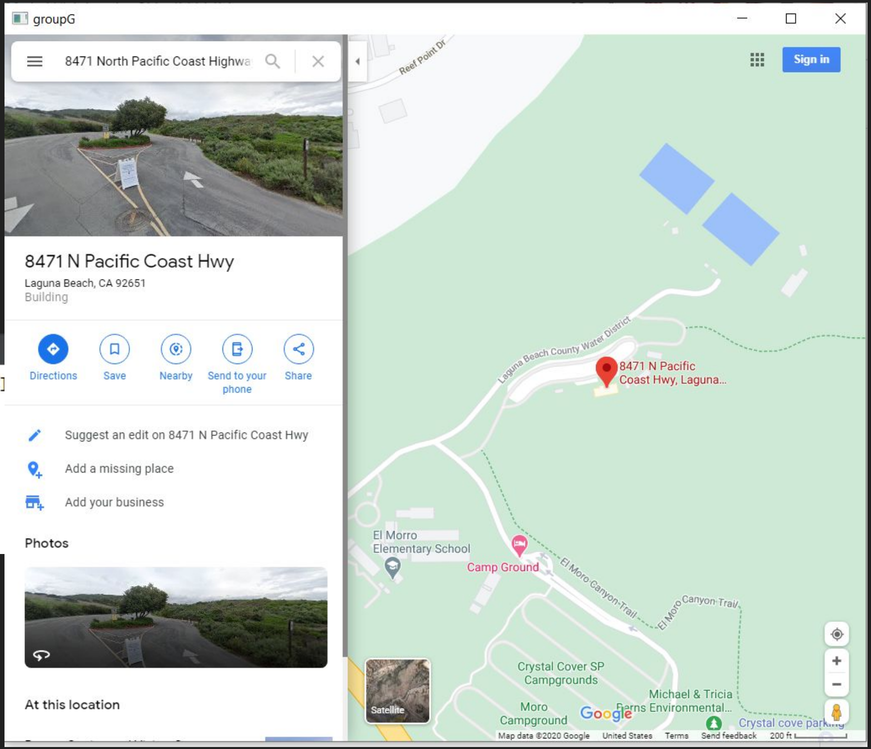
```
QGeoPath dist(position->getPath());
ui->distance->setText(QString::number(toMiles(dist.length(0,dist.size()-1)),'f',2));
```

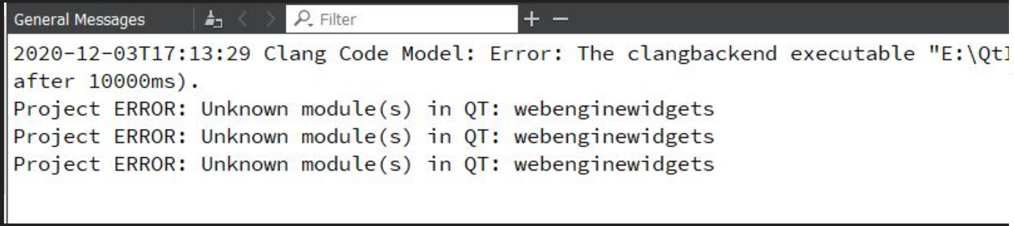


```
double speed = toMiles(dist.length(0,dist.size()-1)) / (time->minute() + time->second()/60.0);
```

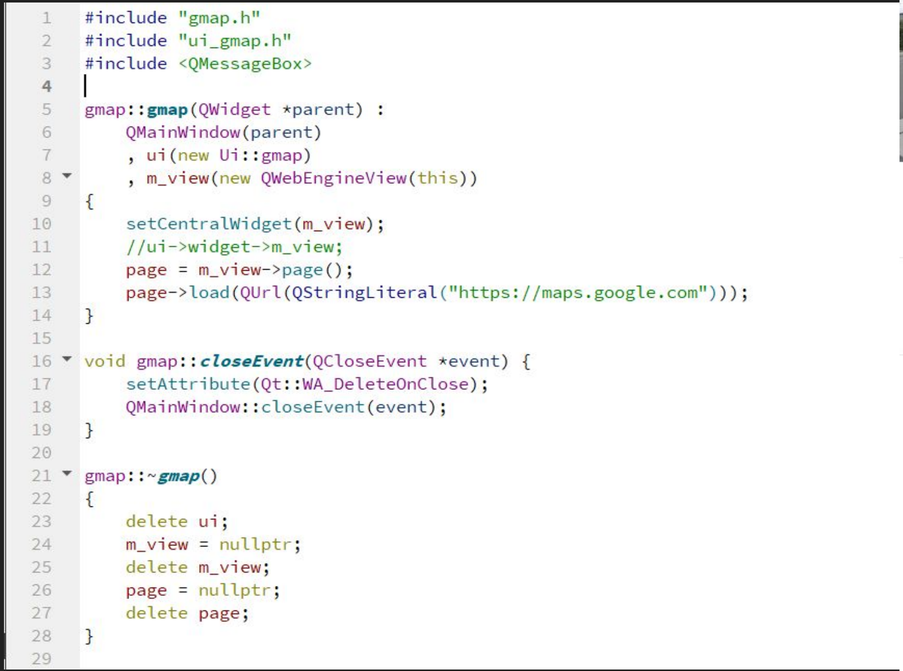
Location

- Displays malleable map
- Uses both QWebEngine and QtLocation to access Google Maps
- Plagued with Administrative Errors
- Actual code is simple

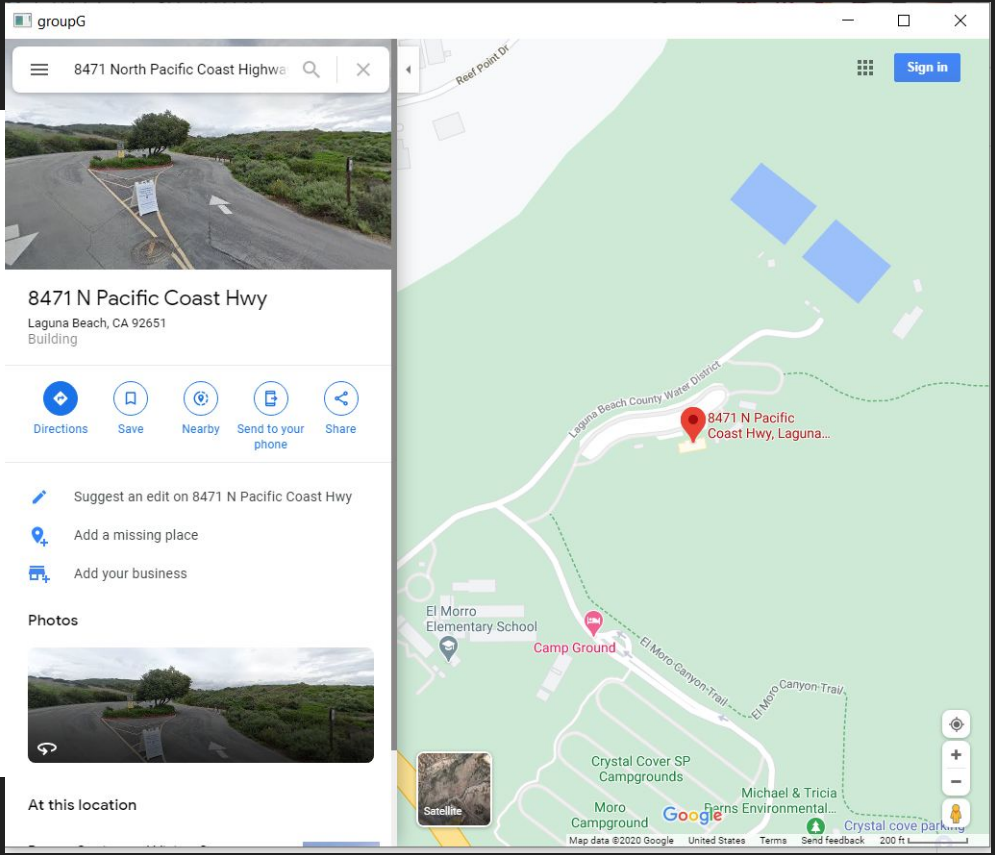




```
2020-12-03T17:13:29 Clang Code Model: Error: The clangbackend executable "E:\Qt]
after 10000ms).
Project ERROR: Unknown module(s) in QT: webenginewidgets
Project ERROR: Unknown module(s) in QT: webenginewidgets
Project ERROR: Unknown module(s) in QT: webenginewidgets
```



```cpp
#include "gmap.h"
#include "ui_gmap.h"
#include <QMessageBox>

gmap::gmap(QWidget *parent) :
    QMainWindow(parent)
    , ui(new Ui::gmap)
    , m_view(new QWebEngineView(this))
{
    setCentralWidget(m_view);
    //ui->widget->m_view;
    page = m_view->page();
    page->load(QUrl(QStringLiteral("https://maps.google.com")));
}

void gmap::closeEvent(QCloseEvent *event) {
    setAttribute(Qt::WA_DeleteOnClose);
    QMainWindow::closeEvent(event);
}

gmap::~gmap()
{
    delete ui;
    m_view = nullptr;
    delete m_view;
    page = nullptr;
    delete page;
}
```



LIVE DEMO