Portfolio Manager

Oliver Larsson

What is the project?

- Automated portfolio manager
- Uses securities from Nasdaq 100, DOW Composite, & top performing ETFs across all indices
- Personalized based on several risk criteria, age, wealth, & investment sector
- Forecasting done using confidence intervals on % change in price

DEMO!

Technologies

- C++
- Plan was to build the text UI first, then transfer that over to Ultralight HTML
- Struggled with just getting feet off the ground for weeks so I decided on QT
- SQLite backend

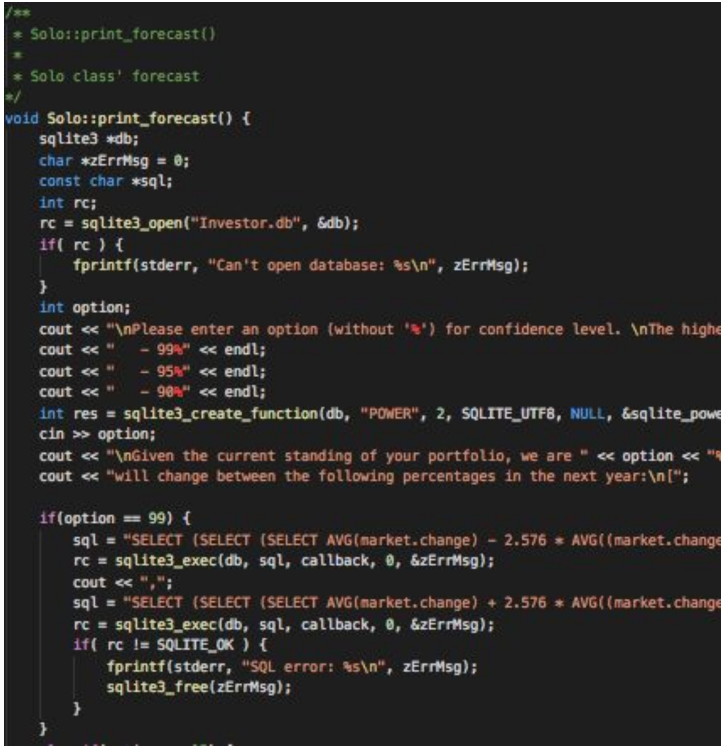
Classes/Views

- Investor
    - Getting information on the user
- Market
    - Takes user’s sector preferences and handles the ‘market’ based on that
- Portfolio
    - Singleton design
    - Tons of background calculations to figure out the portfolio distribution
- Forecasting
    - Factory design
    - Similar to portfolio with algos for forecasting behind the scenes

Major Technical Hurdle

- SQLite
    - Super great for a low-scale projects
    - The ‘Lite’ in SQLite is literal
    - Lacks aggregate SQL functions like STDEV, VAR, SQRT
    - Had to work around the issues separately in text UI and QT

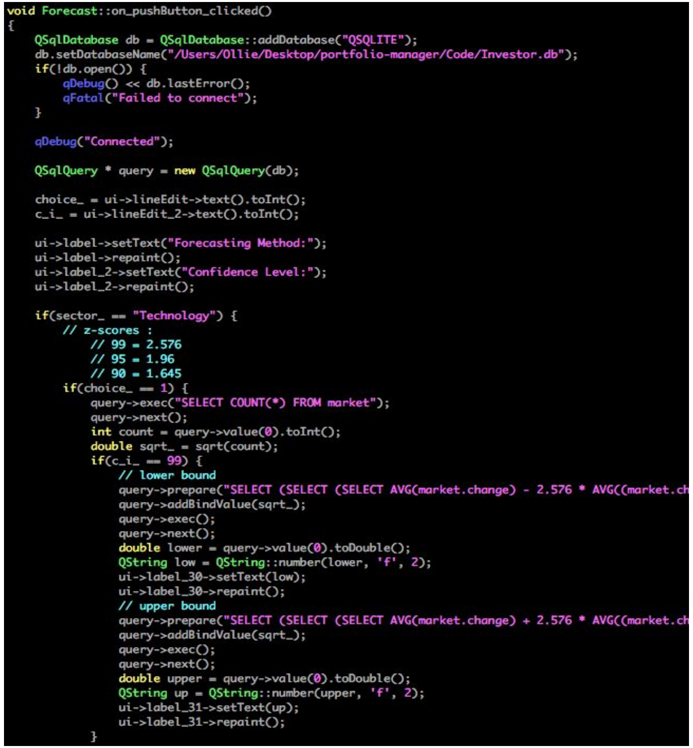
Text UI



```
/**
 * Solo::print_forecast()
 *
 * Solo class' forecast
*/
void Solo::print_forecast() {
    sqlite3 *db;
    char *zErrMsg = 0;
    const char *sql;
    int rc;
    rc = sqlite3_open("Investor.db", &db);
    if( rc ) {
        fprintf(stderr, "Can't open database: %s\n", zErrMsg);
    }
    int option;
    cout << "\nPlease enter an option (without '%') for confidence level. \nThe highe
    cout << "    - 99%" << endl;
    cout << "    - 95%" << endl;
    cout << "    - 90%" << endl;
    int res = sqlite3_create_function(db, "POWER", 2, SQLITE_UTF8, NULL, &sqlite_powe
    cin >> option;
    cout << "\nGiven the current standing of your portfolio, we are " << option << "%
    cout << "will change between the following percentages in the next year:\n[";

    if(option == 99) {
        sql = "SELECT (SELECT (SELECT AVG(market.change) - 2.576 * AVG((market.change
        rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
        cout << ",";
        sql = "SELECT (SELECT (SELECT AVG(market.change) + 2.576 * AVG((market.change
        rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
        if( rc != SQLITE_OK ) {
            fprintf(stderr, "SQL error: %s\n", zErrMsg);
            sqlite3_free(zErrMsg);
        }
    }
```

QT



```
void Forecast::on_pushButton_clicked()
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("/Users/Ollie/Desktop/portfolio-manager/Code/Investor.db");
    if(!db.open()) {
        qDebug() << db.lastError();
        qFatal("Failed to connect");
    }

    qDebug("Connected");

    QSqlQuery * query = new QSqlQuery(db);

    choice_ = ui->lineEdit->text().toInt();
    c_i_ = ui->lineEdit_2->text().toInt();

    ui->label->setText("Forecasting Method:");
    ui->label->repaint();
    ui->label_2->setText("Confidence Level:");
    ui->label_2->repaint();

    if(sector_ == "Technology") {
        // z-scores :
            // 99 = 2.576
            // 95 = 1.96
            // 90 = 1.645
        if(choice_ == 1) {
            query->exec("SELECT COUNT(*) FROM market");
            query->next();
            int count = query->value(0).toInt();
            double sqrt_ = sqrt(count);
            if(c_i_ == 99) {
                // lower bound
                query->prepare("SELECT (SELECT (SELECT AVG(market.change) - 2.576 * AVG((market.ch
                query->addBindValue(sqrt_);
                query->exec();
                query->next();
                double lower = query->value(0).toDouble();
                QString low = QString::number(lower, 'f', 2);
                ui->label_30->setText(low);
                ui->label_30->repaint();
                // upper bound
                query->prepare("SELECT (SELECT (SELECT AVG(market.change) + 2.576 * AVG((market.ch
                query->addBindValue(sqrt_);
                query->exec();
                query->next();
                double upper = query->value(0).toDouble();
                QString up = QString::number(upper, 'f', 2);
                ui->label_31->setText(up);
                ui->label_31->repaint();
            }
```

Questions?