Frayda Zirkind

TaskBadger Project Documentation



# Executive Summary

TaskBadger is a personal project management desktop application that organizes and tracks projects and their related tasks. Projects can be created and broken down into smaller tasks that need to be done to complete the project. Projects and tasks can be assigned priorities and deadlines. Statuses can be updated as the user progresses through each task and project. Additionally, lists can be generated and associated with a project or task.

TaskBadger is a data based application that will store all information in a local, secure database. This data can be retrieved every time the user re-opens TaskBadger. The user can create new entries, edit or update existing entries or delete entries using this application.

TaskBadger is designed for the user who wants to organize their various projects in one place. The user may already use other applications for tracking projects, but TaskBadger can be used to consolidate all those scattered projects. Some users may not use any project management application because they seem too intimidating or overwhelming to even begin to use. TaskBadger solves these problems by providing a user-friendly centralized platform that can be used for all kinds of projects. This application is designed with the end-user in mind.

TaskBadger is written in C++ and uses a MySQL database. It utilizes inheritance, polymorphism, and abstract classes to make the code re-usable and highly maintainable.

TaskBadger is available on GitHub:

GitHub Repository: <https://github.com/fraydaz/TaskBadger>

GitHub Project Page: <https://github.com/fraydaz/TaskBadger/tree/master/>

GitHub Project Documentation: <https://github.com/fraydaz/TaskBadger/tree/master/doc>

General walkthrough video: <https://drive.google.com/file/d/1jJelSEI3q0SSfOdfSFLvJl7Z5krnYeE6/view?usp=sharing>

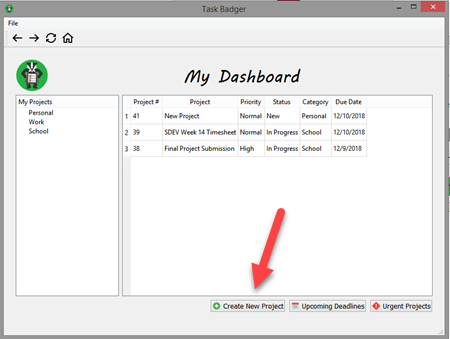
Database walkthrough video: <https://drive.google.com/file/d/1XAhcmkXSDNLaaYGxQOcRmIkLkEml_wlr/view?usp=sharing>

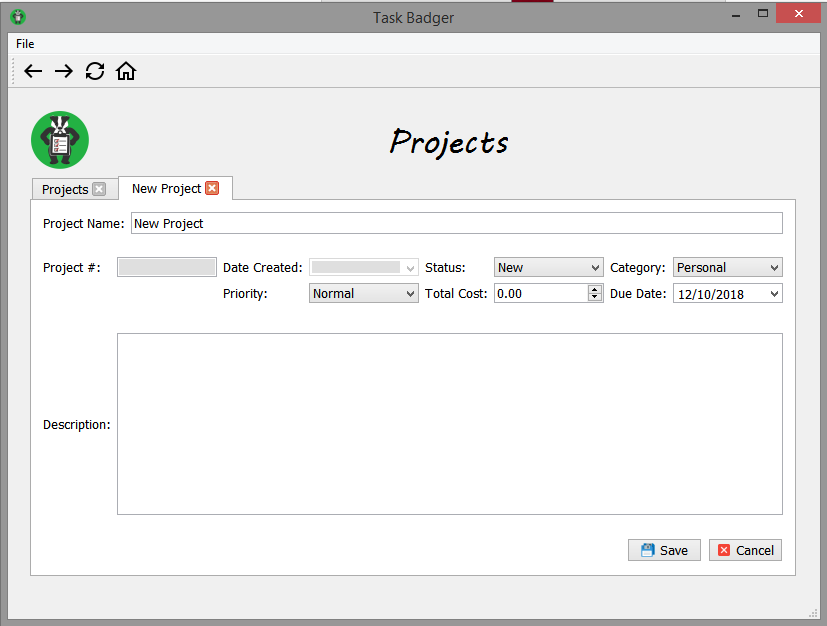
Reusable code walkthrough video: <https://drive.google.com/file/d/149i9wl3UPez59QEL8njQcZB33NXwWJNl/view?usp=sharing>

Verification and validation walkthrough video: <https://drive.google.com/file/d/1rvcV6Y1QhdRknWv57eUUsgJOhdqmG3HT/view?usp=sharing>

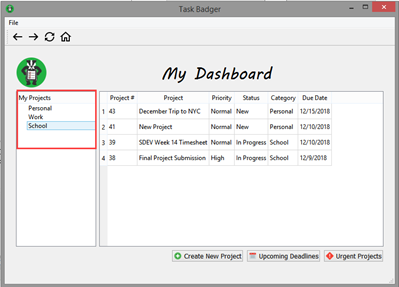
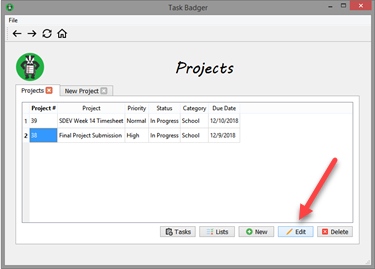
Project Completion Status: 100%

# Scenario | Creating a New Project

The user launches the program and the home page loads. The user selects the **Create New Project** button at the bottom of the page. The application opens the **New Project** form. The user enters the project information and then saves the new project.

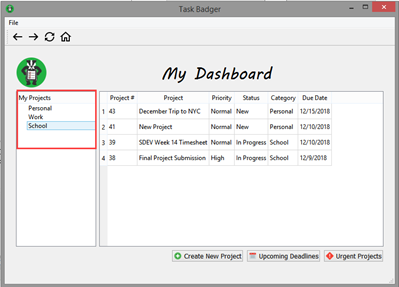
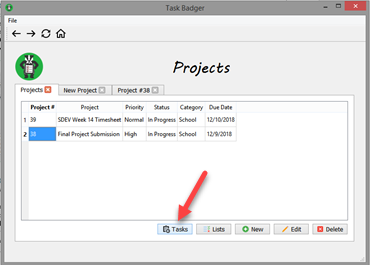
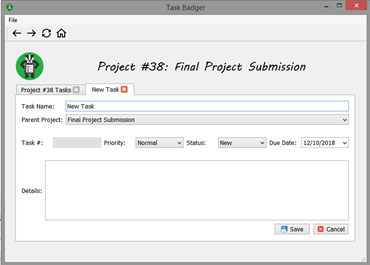
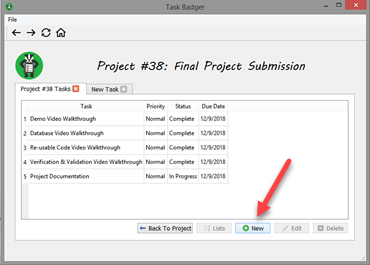


# Scenario | Opening an Existing Project

The user launches the program and the home page loads. The user selects the project category from the menu on the left side. The projects page loads and displays all projects that belong to the selected category. The user selects the project from the list and clicks the **Edit** button. The project opens in a new tab for the user to view and edit.

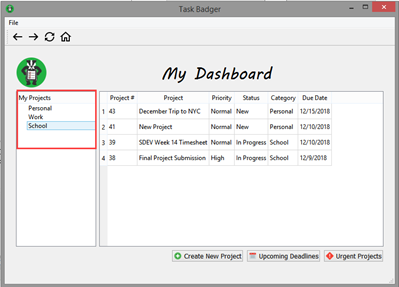
# 

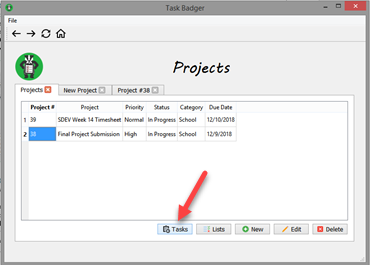
# Scenario | Creating a New Task for a Project

The user launches the program and the home page loads. The user selects the project category from the menu on the left side. The projects page loads and displays all projects that belong to the selected category. The user selects the project from the list and clicks the **Tasks** button. The Tasks page loads with all tasks for the selected project. The user selects the **New** button. The application opens the **New Task** form. The user enters the task information and then saves the new task.

# Scenario | Opening an Existing Task

The user launches the program and the home page loads. The user selects the project category from the menu on the left side. The projects page loads and displays all projects that belong to the selected category. The user selects the project from the list and clicks the **Tasks** button. The Tasks page loads with all tasks for the selected project. The user selects the task from the list and clicks the **Edit** button. The task opens in a new tab for the user to view and edit.



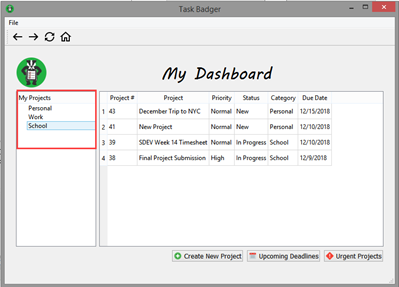


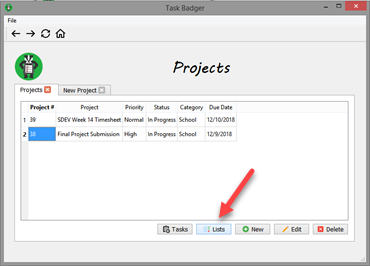
# 

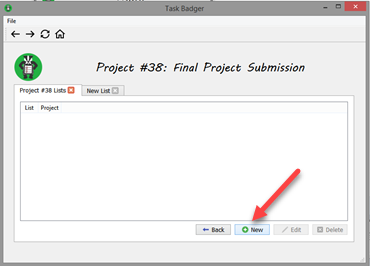
# 

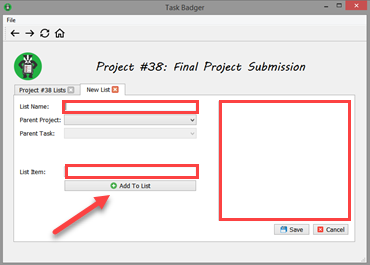
# Scenario | Creating a New List for a Project

The user launches the program and the home page loads. The user selects the project category from the menu on the left side. The projects page loads and displays all projects that belong to the selected category. The user selects the project from the list and clicks the **Lists** button. The Lists page loads showing all lists for the selected project. The user selects the **New** button. The application opens the **New List** form. The user enters the list information and then saves the new list. The user can add items in the **List Item** field and then click the **Add To List** button. The application reloads the list view displaying the list items.



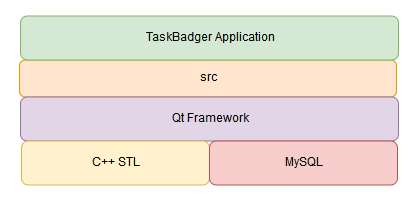






# System Architecture

TaskBadger runs as a single-user, local application on Microsoft Windows. The application code is all contained in the executable file.



## Source Code Structure

Source code is all contained in the src directory. The SQL file to create the database initially is located in the doc directory. The following is a summary of the source code directories and their contents:

|  |  |
| --- | --- |
| **Code Directory** | |
| **Directory** | **Usage** |
| / | Project root directory . Contains all other directories and the readme file. |
| /deploy | Contains the project executable file with all its dependencies |
| /deploy/imageformats | Contains the application extensions for image formatting required to run TaskBadger. |
| /deploy/platforms | Contains the application extension required to run TaskBadger on the Windows platform. |
| /deploy/sqldrivers | Contains the application extensions for connecting the application to MySQL. These extensions are required to run TaskBadger. |
| /deploy/styles | Contains the application extensions for formatting the application window. This is required to run TaskBadger. |
| /doc | Contains project documentation files and SQL source file to create database. |
| /images | Contains all the images and icons used in TaskBadger. |
| /release | Contains project files generated by Qt Creator. |
| /src | Contains source and header files for the application, as well as the Qt project file, make file and layout files generated by Qt. |
| *Highlighted rows indicate directories containing source code.* | |

# Executables

### TaskBadger (TaskBadger.exe)

This is the only application executable that runs the program.

Code Architecture

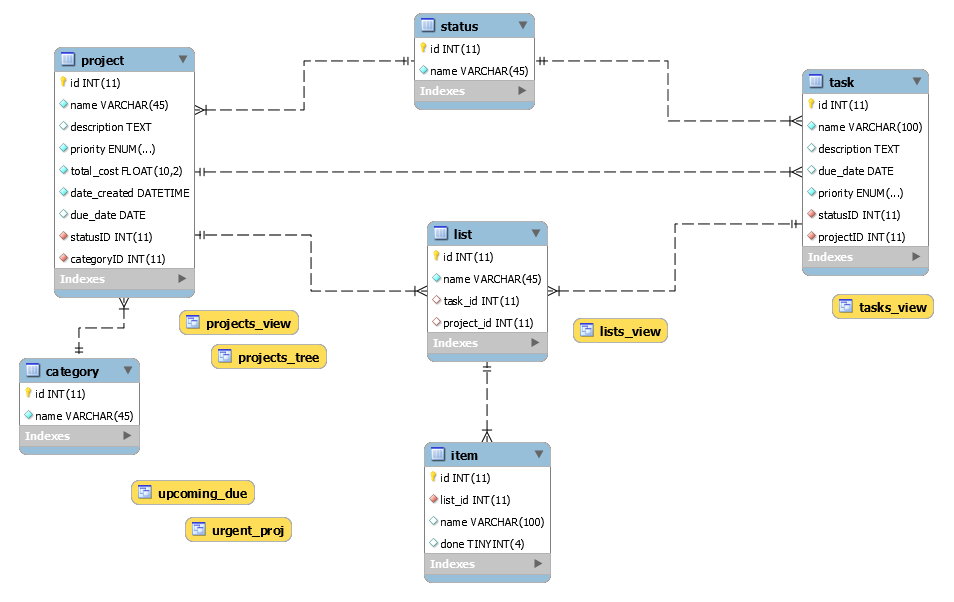
The application interface centers around the MainWindow class, which contains functionality for most of the layout.

The database consists of four main tables with two helper tables and several view and procedures. The main tables include project, task, list, and item. The helper tables are status and category. The views and stored procedures keep as much database logic out of the code as possible. All views and tables are saved as a database view or stored procedure in the database.

Object classes (i.e. Project, Task, List, and ListItem) inherit functionality from the DBObject class. The DBObject class stores data about the object and contains get and set functions. The object classes each contains a pointer to their respective DAO class (i.e. ProjectDAO, TaskDAO, ListDAO, and ItemDAO). The Data Access Object classes all inherit functionality from the ObjectDAO class. These classes act as an interface to separate database logic and queries from application logic. Model classes (i.e. ProjectModel, TaskModel, and ListModel) inherit functionality from the ObjectModel class. The model classes each contains a pointer to their respective DAO class to get data from the database for setting up the layouts. The FormLayout classes (i.e. ProjectFormLayout, TaskFormLayout, and ListFormLayout) inherit functionality from the abstract FormLayout class. These classes create a dynamic layout when an existing project, task, or list is opened in a new tab.

# Database or Data Store

All the data is stored in a MySQL database. The table structure is diagrammed below. The database views are included in the diagram in yellow bubbles.



## Views, Stored Procedures and User Defined Functions

The projects\_view view shows all projects with renamed column headers. This view also shows the status and category names. The tasks\_view shows all tasks with renamed column headers. This view shows status and project names. The lists\_view shows all lists with renamed column headers and with the task / project name. The projects\_tree view shows all the category names only for the tree view on the application home page. The upcoming\_due view shows all projects due today or later and in order of due date, with earliest due date first. This view is used on the ‘upcoming deadlines’ page. The urgent\_proj view shows all projects with ‘high’ or ‘highest’ priority, showing highest priority first.

There are five stored procedures used in this project:

1. List\_items: shows all items that belong to the list id passed as a parameter
2. Project\_category: shows all projects that have the category id passed as a parameter
3. Project\_lists: shows all lists that have the project id passed as a parameter
4. Project\_tasks: shows all tasks that have the project id passed as a parameter
5. Task\_lists: shows all lists that have the task id passed as a parameter

Programming Language | C++

TaskBadger is written in C++ and uses version 5.11.1 of the Qt framework. This project utilizes a number of the Qt library classes. On the Windows platform, this project requires the following DLLs from the Qt framework:

* libgcc\_s\_dw2-1.dll
* libstdc++-6.dll
* libwinpthread-1.dll
* Qt5Core.dll
* Qt5Gui.dll
* Qt5Sql.dll
* Qt5Svg.dll
* Qt5Widgets.dll
* qgif.dll
* qicns.dll
* qico.dll
* qjpeg.dll
* qsvg.ll
* qtga.dll
* qtiff.dll
* qwbmp.dll
* qwebp.dll
* qwindows.dll
* qsqlmysql.dll
* qwindowsvistastyle.dll

Project Classes

Classes within the project are used to abstract re-usable pieces of code. Classes are also used to group related values, known as properties. The project utilizes these classes:

### DBManager | DBManager.h, DBManager.cpp

Connects to database and handles all database queries

### DBObject | DBObject.h, DBObject.cpp

Base class for all objects with a database table

### Exception | Exception.h, Exception.cpp

Shows dialog boxes with success / failure messages from SQL queries

### FormLayout | FormLayout.h, FormLayout.cpp

Abstract base class for all classes that set up the UI dynamically

### ItemDAO | ItemDAO.h, ItemDAO.cpp

Inherits from ObjectDAO class and acts as an interface between list items and the database

### List | List.h, List.cpp

Inherits from DBObject class and implements functionality for lists

### ListDAO | ListDAO.h, ListDAO.cpp

Inherits from ObjectDAO class and acts as an interface between lists and the database

### ListFormLayout | ListFormLayout.h, ListFormLayout.cpp

Inherits from FormLayout class and sets up the UI dynamically for editing lists in a new tab

### ListItem | ListItem.h, ListItem.cpp

Inherits from DBObject class and implements functionality for list items

### ListModel | ListModel.h, ListModel.cpp

Inherits from the ObjectModel class and sets up the list layouts with data from the database

### MainWindow | MainWindow.h, MainWindow.cpp

Implements the GUI interface window

### ObjectDAO | ObjectDAO.h, ObjectDAO.cpp

Base class for all DAO classes, acting as an interface between objects and the database

### ObjectModel | ObjectModel.h, ObjectModel.cpp

Base class for all model classes to set up the layouts with data from the database

### Project | Project.h, Project.cpp

Inherits from DBObject class and implements functionality for projects

### ProjectDAO | ProjectDAO.h, ProjectDAO.cpp

Inherits from ObjectDAO class and acts as an interface between projects and the database

### ProjectFormLayout | ProjectFormLayout.h, ProjectFormLayout.cpp

Inherits from FormLayout class and sets up the UI dynamically for editing projects in a new tab

### ProjectModel | ProjectModel.h, ProjectModel.cpp

Inherits from the ObjectModel class and sets up the project layouts with data from the database

### Task | Task.h, Task.cpp

Inherits from DBObject class and implements functionality for tasks

### TaskDAO | TaskDAO.h, TaskDAO.cpp

Inherits from ObjectDAO class and acts as an interface between tasks and the database

### TaskFormLayout | TaskFormLayout.h, TaskFormLayout.cpp

Inherits from FormLayout class and sets up the UI dynamically for editing tasks in a new tab

### TaskModel | TaskModel.h, TaskModel.cpp

Inherits from the ObjectModel class and sets up the task layouts with data from the database

Summary

TaskBadger is a project management desktop application written in C++ using the Qt library for a user-friendly interface. The data from this application is stored in a MySQL database. The program is a local application designed for Windows OS, though it can be easily ported to Linux by adding the appropriate application extensions.

Creating this project requires connecting the MySQL database to the C++ code that drives the program’s functionality. The GUI has been designed using Qt Creator and the interface is linked to the C++ code for functionality.

The current version of this application consists of a basic project management program which can be used to create project and break each project down into smaller tasks. Lists can be created and linked to a project or task. Different views of the projects allow users to quickly track the most urgent unfinished projects as well as projects with deadlines coming up soonest. Projects can be categorized for better organization and management.

# APPENDIX A (TEST PLAN)

TaskBadger has been tested thorough using user scenarios. For each scenario, the tester runs the program and follows the required steps to complete each scenario. The output in the debugger and the test results are manually verified for accuracy. Any incorrect or unexpected result is considered a bug. Some scenarios that were tested include:

* Creating and saving a new project
* Editing an existing project
* Adding task to a project
* Creating a list for a project
* Deleting a project (with tasks and without tasks)
* Creating a task
* Updating an existing task
* Deleting a task
* Creating a new list for a task
* Adding items to a new list
* Updating an existing list with new items
* Attempt to delete a task that has a list

# APPENDIX B (BUILD AND RELEASE PROCESS)

The following steps are required to update TaskBadger:

1. Checkout the master branch on GitHub (<https://github.com/fraydaz/TaskBadger>).
2. Apply fixes in Qt Creator.
3. Build the project in “release” mode.
4. Copy the executable file from the “Release” directory into the “deploy” directory, replacing the existing TaskBadger executable file.
5. Add any application extensions required for the updated version to the “deploy” directory.
6. Distribute the updated “deploy” directory to users.

# APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)

The following steps are required to run TaskBadger on a desktop:

1. Install WAMP Server (<https://sourceforge.net/projects/wampserver/>)
2. Download the MySQL database (<https://github.com/fraydaz/TaskBadger/blob/master/doc/task_badger.sql>)
3. Import the database in PhpMyAdmin
4. Create a user account for the TaskBadger database with the username: badger and password: champlain
5. Download the “deploy” directory from GitHub (<https://github.com/fraydaz/TaskBadger>)
6. Run TaskBadger.exe from the deploy directory

# APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)

The following steps are required to properly configure a development environment to work on TaskBadger:

1. Install Qt Framework v5.11.1 (open source). (https://www.qt.io/download-open-source/)
2. Download the Qt Source Code (ex: qt-everywhere-opensource-src-5.11.1.zip)
3. Extract the Qt Source Code into the Qt folder
4. Copy qsqlmysql.dll and qsqlmysqld.dll from C:\Qt\Qt5.0.1\Sources\qtbase\plugins\sqldrivers to C:\Qt\Qt5.0.1\5.0.1\mingw47\_32\plugins\sqldrivers.
5. Install MySQL Community Server (ex: mysql-5.5.30-win32.msi)
6. Copy libmysql.dll from the MySQL directory to C:\Windows
7. Install WAMP Server (<https://sourceforge.net/projects/wampserver/>)
8. Download the MySQL database (<https://github.com/fraydaz/TaskBadger/blob/master/doc/task_badger.sql>)
9. Import the database in PhpMyAdmin
10. Clone the project from GitHub (<https://github.com/fraydaz/TaskBadger>)