# Black Journal

Design Document

Table of Contents

[1 Introduction 3](#__RefHeading___Toc195405481)

[1.1 Purpose and Scope 3](#__RefHeading___Toc195405482)

[1.2 Target Audience 3](#__RefHeading___Toc195405483)

[1.3 Terms and Definitions 3](#__RefHeading___Toc195405485)

[2 Design Considerations 4](#__RefHeading___Toc195405486)

[2.1 Constraints and Dependencies 4](#__RefHeading___Toc195405488)

[2.2 Methodology 4](#__RefHeading___Toc195405489)

[3 System Overview 5](#__RefHeading___Toc195405490)

[4 System Architecture 6](#__RefHeading___Toc195405491)

[4.1 Users 6](#__RefHeading___Toc195405492)

[4.1.1 User model 6](#__RefHeading___Toc195405493)

[4.1.2 User controller 6](#__RefHeading___Toc195405494)

[4.1.3 User view 6](#__RefHeading___Toc412_249614142)

[4.2 Tasks & Task Lists 7](#__RefHeading___Toc195405495)

[4.2.1 Task Model 7](#__RefHeading___Toc414_249614142)

[4.2.2 Task List Model 7](#__RefHeading___Toc416_249614142)

[4.2.3 Task Controller 7](#__RefHeading___Toc418_249614142)

[4.2.4 Task View 7](#__RefHeading___Toc420_249614142)

[5 Detailed System Design 10](#__RefHeading___Toc195405497)

[5.1 Users 10](#__RefHeading___Toc195405498)

[5.1.1 User 10](#__RefHeading___Toc195405499)

[5.1.2 UserView 10](#__RefHeading___Toc195405500)

[5.1.3 UserController 11](#__RefHeading___Toc422_249614142)

[5.2 Tasks & Task Lists 11](#__RefHeading___Toc195405501)

[5.2.1 Task 11](#__RefHeading___Toc424_249614142)

[5.2.2 TaskList 11](#__RefHeading___Toc426_249614142)

[5.2.3 TaskView 12](#__RefHeading___Toc428_249614142)

[5.2.4 TaskController 12](#__RefHeading___Toc430_249614142)

# Introduction

This document describes the architecture, system, and design of the Black Journal Task Management Application. I will be going into detail as to what kind of design methodologies are used, as well as the constraints and requirements needed before implementing the software. In Sections 3 and 4, I will introduce the project from a high-level viewpoint, and in section 5 I will go into much detail about the specific components of the application.

## Purpose and Scope

The purpose of this document is to provide the client with a detailed and comprehensive design specification for the project.

## Target Audience

Black Journal is designed to provide quick and easy task management functionality for the busy university student. Frequent changes in schedule, tasks to be completed, and frequent deadlines are the problems Black Journal hopes to make less cumbersome.

## Terms and Definitions

* **Black Journal** – Project name
* **Black Journal Task Management Application** – Verbose name
* **User** – The class defining the end user's account
* **Task List** – The class defining where tasks are stored
* **Task –** The class defining individual task objects
* **Model** – The component that represents an entity class
* **View –** The component defining how information is presented to the username
* **Controller –** The component that interacts and operates between the view and model.

# Design Considerations

This section introduces the initial state going into the project before development starts. I will introduce the functional and non-functional requirements that are needed to build the application properly, as well as the software methodologies used going in to develop the project.

## Constraints and Dependencies

Functional requirements of Black Journal include a User Management System to allow users to create accounts so their task lists are unique and persistent. The application also requires Tasks and Task Lists to manage the tasks. Some other functional requirements include controllers and views for each class type.  
Non-functional requirements for Black Journal include using the Java language, as well as a MySQL Database Management System.

## Methodology

The software methodology I am using is a Structured software methodology. In this methodology, I create small but critical parts of the application first (according to requirements and design), and slowly build off of it in order to detect early faults or bugs that can be fixed over a short time-span, and before the complexity grows too much. I am also specifically using a model-view-controller design paradigm for this application for ease of development and use, which allows me to separate the interface from the model entity and the controls between them.

# System Overview

The Black Journal application is comprised of three primary models: Users, Task Lists, and Tasks. Each model has a corresponding view that is used to present information to the end user, and a controller that updates the view and model as needed. End users can log in and out of user accounts, as well as create, update, view, and delete their account if they wish. Once a user has an account, the user can create, update, view, and delete tasks in their respective task lists. There will only be 3 task lists by default: “TODO”, “IN PROGRESS”, and “DONE”. Users can view all the tasks, clear all tasks from a list, and move tasks from one list to another. Create, detail view, update, and delete operations will be delegated to the task objects.

# System Architecture

Users, Tasks, and Task Lists make up this application. Users have three task lists (TODO, IN PROGRESS, DONE), and once they have logged in, they are able to see the tasks within the task lists.

## Users

The User module in this application is responsible for keeping track of a User's tasks within their private Task Lists. Each user is required to create a user and log in to that user in order to view their personalized task lists. Users can be created, edited, viewed, and deleted as well as logged in and out of the system.

### User model

The User model acts as an entity class for the users of the system. Users have an id assigned to them, as well as a chosen username and password. Users can be created, edited, and deleted.

### User controller

The User Controller acts as a control class for the User model, as well as the User view. The user controller will perform validation on incoming data from the User view, and is responsible for updating the User model and the User view. User controller will have the ability to set and get parameters to and from the User model, as well as authentication.

### User view

The User view acts as a boundary class for the User model, is responsible for displaying user information on the screen. The User view is controlled by the User controller, and provides the interface for logging users in and out of the system, as well as giving users the option to create a new user and edit/delete an existing user that they control. This will accept input from the user and pass it to the User controller for data validation.

## Tasks & Task Lists

Tasks are objects that represent an item to be done. Multiple Tasks can live within a Task List. There will be 3 task lists for the user to add to, remove from, move, or edit tasks within. The Task View is responsible for displaying tasks within task lists to the user, as well as taking User data when a task must be updated. The Task Controller will take this data and perform the necessary operations to create, edit, view, and delete tasks, as well as move tasks to new lists.

### Task Model

The Task model acts as an entity class for storing Task data. Each task has an id, name, description, and belongs to a Task List. Tasks have the ability to set and get all of it's fields.

### Task List Model

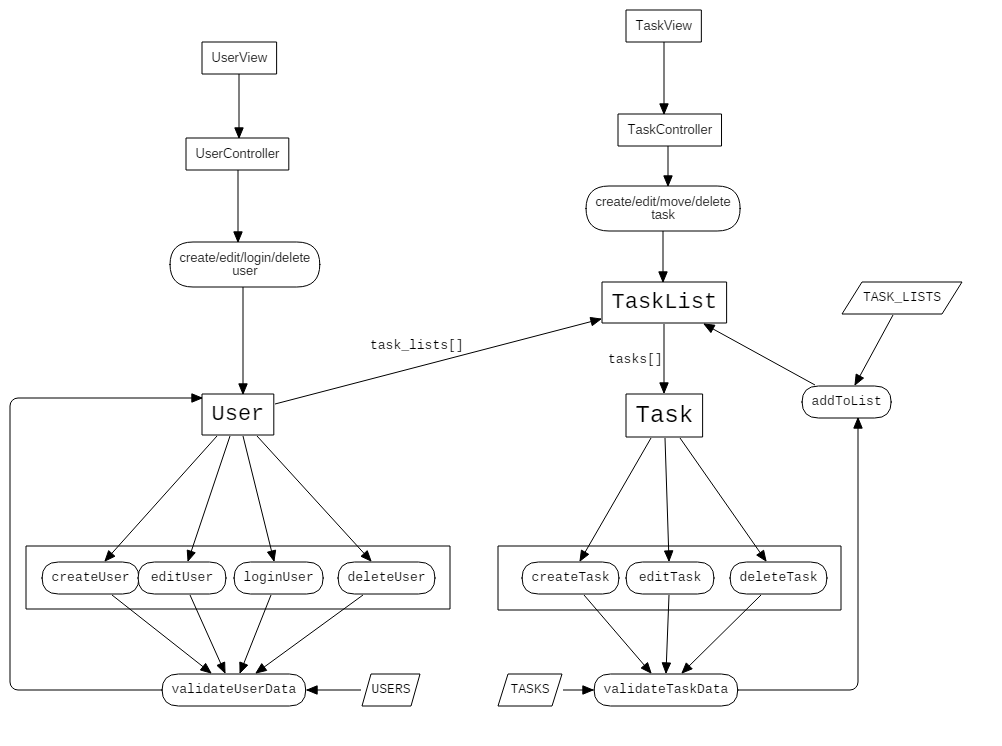
The Task List model acts as an entity class for storing Tasks in one of three objects (TODO, IN PROGRESS, DONE). Each Task List has it's own id and title, and is responsible for setting and getting initial data, as well as retrieving all it's tasks and adding to the task list.

### Task Controller

The Task controller is one of the larger classes in the application. The Task controller is responsible for creating and updating Task model data, as well as updating the Task view to any changes from the model. The 3 Task Lists will contain all tasks in the application, and the Task controller is responsible for moving tasks from one Task List to another. New tasks to be created will be created in one of the three lists available to the user (TODO, by default).

### Task View

The Task view acts as a boundary class for both Task Lists and Tasks. Users can submit data here that will be passed to the Task controller if they wish to create a new Task. Data validation will be performed in the Task controller. New Task Lists will not be created by the user, but the user will have the option to move the Task to a new list (i.e., moved to the “IN PROGRESS” or “DONE” queue).

State Chart Diagram for Black Journal

# Detailed System Design

The Black Journal application is comprised of Users, Tasks, and Task Lists as well as their respective controllers and views. Using this model-view-controller paradigm, the control, interface, and entities are properly separated and interact with each other efficiently. Views take user input and actions, and pass user input and actions to the controller. The controller dictates what to perform with that input or action, as well as updating the model with the appropriate data, and updating the view when the model is updated.

## Users

The user module for this application is comprised of three classes: The User class, which is an entity for storing user data and getting/setting it's data. The UserView class, which is responsible for taking user input/actions and relaying them to the UserController class. The UserController class is responsible for performing create/read/update/delete operations on the user, as well as validating data and authenticating the user upon login.

### User

The User model will be identified via a unique user\_id. Each user will have a username and a password fields, as well as an array task\_lists[], which contain the users' private TaskLists.. Setters and getters will be provided. When a User is created, it is required that the constructor accepts a valid username and password (input validation is handled by the UserController class). As a part of the MVC design paradigm, all the model will do is store and manage it's own data.

### UserView

The UserView class will take input data and pass it to the UserController, where further processing will be done. The UserView will also be updated by the UserController. The class's purpose is to act as an interface for the user to modify user objects, as well as log in and out.

### UserController

The UserController contains both the UserView and the UserModel. When passed input data from the view, this will validate that data and, upon successful validation, update the appropriate fields in the User model, and update the view after. When a user requests to log in, it will also validate that data against an existing User model, and will then provide access to the TaskView containing that users' TaskLists & Tasks. All User data will be stored in a table in a MySQL database.

## Tasks & Task Lists

The four classes defined below work together to supply the Task Management functionality for the application. Task and TaskList serve as model entities to store task information and tasks themselves (respectively). TaskView is responsible for getting user input and relaying it to the TaskController. The TaskController class is a control class that performs the necessary operations to create, update, read, and delete tasks, as well as get all the tasks from the list and move a task to another list.

### Task

The Task class contains parameters task\_id, name, and description. Each Task object lives inside one of the 3 TaskList objects that will be created. A single Task object has access to only it's data, but it can be retrieved from a tasks[] array from it's respective TaskList. It's only purpose is to store and set/get it's own data. The Task actual data will live within it's own table in a MySQL database.

### TaskList

The TaskList class contains the parameters task\_list\_id, title, and a tasks[] array. New tasks will be added to the array via the Task controller. This model is responsible only for setting and getting it's private data. This will live within it's own table in a MySQL database. Tasks can be removed from the list, and the array will be adjusted to accommodate this change.

### TaskView

The TaskView will accept user input and pass it to the TaskController class. This class will mainly act as an interface and view for the user to interact with. It will by default list all tasks after the user has logged in, and will be able to add/edit/delete tasks within a task list.

### TaskController

The TaskController class contains references to both the User model and UserView view. It is responsible for validating data given by the user, updating the appropriate model fields (responsible for the creation, updates, and deletion of Tasks), and moving/removing tasks from lists.

Class diagram for the Black Journal application