Amanda Ryman

Term Project: Tudu Task Management Software

Design Document

Table of Contents

1 Introduction

This document will outline the functionality and design framework for the Tudu software product, including limitations, architecture, and an in-depth system design layout.

1.1 Purpose and Scope

This document aims to provide a detailed description of the requirements for the Tudu software, outlining the needs of the users as well as any other restrictions on the program. This document also serves as a proposal to the client so that project direction can be affirmed and tailored to the client's needs.

Tudu is intended for use by the general public as a simple organizational tool. The native app will be available as a free download with possibility for development as a cross-platform mobile web app.

Users will manually add items to their "to do" list with the availability to flag their status and mark them as complete. Beyond the initial download, no further connectivity or input from other programs is necessary, although the addition of online accounts may be supported in future iterations of the project. For example, in future developments, a user will be able to sign into their Tudu account from a public computer and be able to access their information using the web version of the software. Further development may also include support for other user-specified categories.

1.2 Target Audience

This document is provided as a reference document for the current and future development team, and for the client commissioning the development of this program.

2 Design Considerations

This section will provide a brief overview of limitations and considerations for development of the design for the Tudu software.

2.1 Constraints and Dependencies

Privacy is a large component of any software product and all measures will be taken to ensure encapsulation and data integrity. Users will not be able to access other users' accounts. Encapsulation is also necessary to ensure that one user's data does not become entangled with another's.

2.2 Methodology

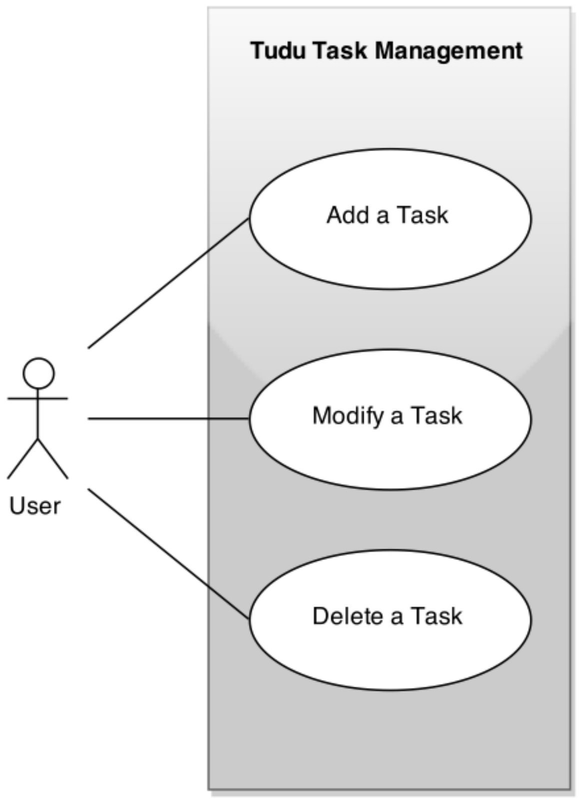
As this project is comprised of clearly bounded entities which interact with each other, an object-oriented approach makes sense. While a simple functional programming approach would have also been appropriate given the simple set of data, I appreciate the modularity of OOP and the inherent ability to easily expand and add complexity to projects down the line. To meet these needs, the OOP design will be implemented in Java.

3 System Overview

This section will illustrate the overall model on which the Tudu software will operate. Tudu supports multiple users. Once a user is logged in, their existing tasks are displayed and they are able to select from various options: add a new task, modify an existing task, and delete a task. The user is able to log out or quit at any time which prompts their data to be saved.

A use case analysis of the three primary tasks is fairly straightforward, as Tudu interacts with a single user and does not collaborate with any other software products.

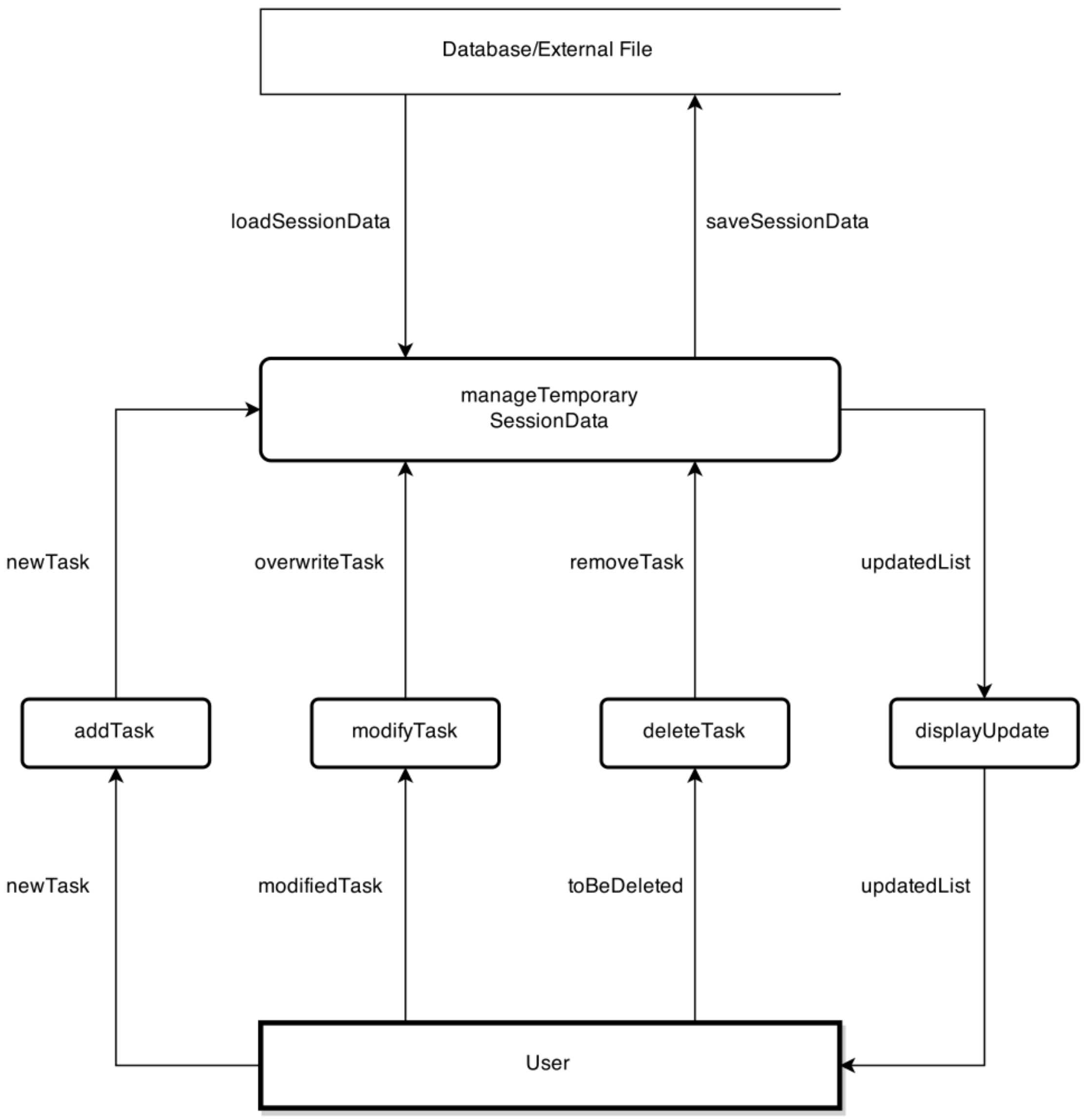
USE CASE DIAGRAM FOR TUDU



4 System Architecture

This section will outline a more in-depth process for the flow of data within Tudu. As mentioned there is only one single user per instance of Tudu at a time. The data is managed from an external database which is read in, modified and then written out at the close of the program. Ignoring the physical data flow, we will focus on the logical data flow within Tudu:

DATA FLOW DIAGRAM FOR TUDU SOFTWARE



4.1 Manage Temporary List Data

This object is synonymous with what will become our “Manager” class. This object is responsible for loading data from the external source, handling it during the session, updating the output to the user as data is modified, and writing out the data at the end of the session. This class is also responsible for validating user access.

4.2 Add a Task

This object is responsible for creating a new Task object and adding it to the temporary list. A new Task consists of a string for the description, a timestamp of when it was created, and is flagged as “To Do” by default.

4.3 Modify a Task

This object is responsible for modifying a Task object and overwriting it in the temporary list. Modifications may consist of a description edit or a flag change (a task has been marked as “in progress” rather than “to do”).

4.4 Delete a Task

This object is responsible for removing a chosen Task object from the temporary list.

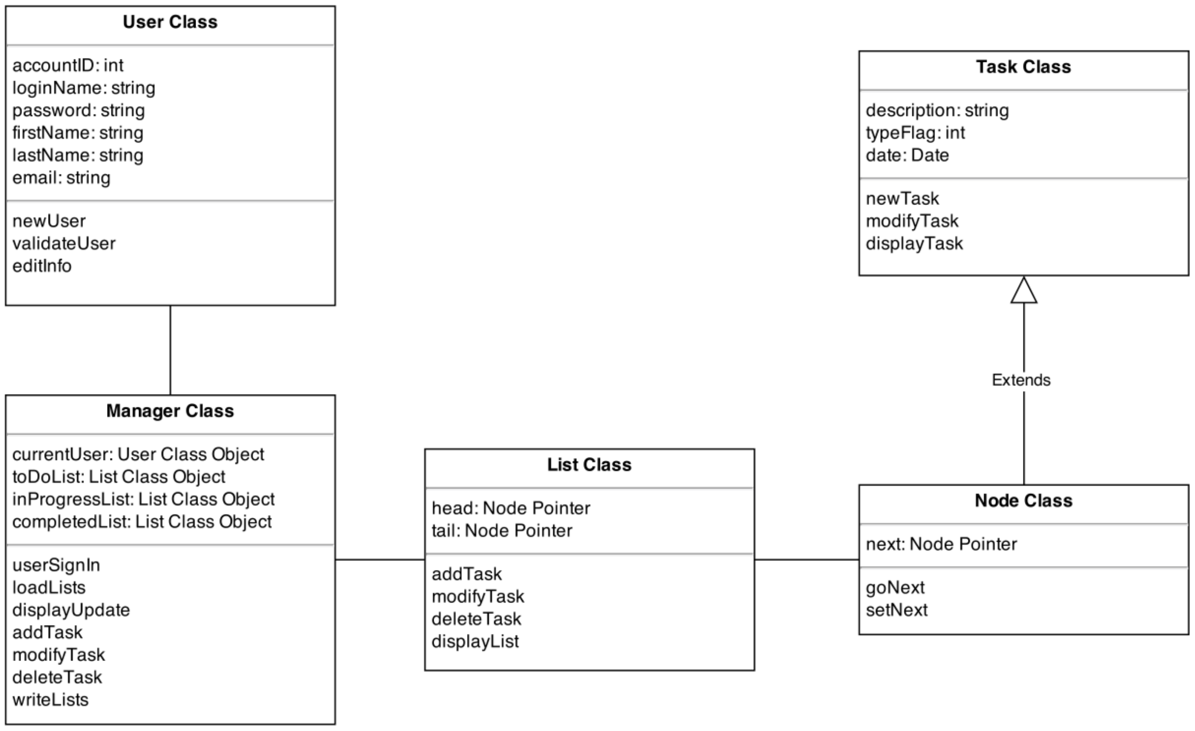
4.5 Update Display

This object is responsible for updating the output to the user (the display) every time a modification is made.

5 Detailed System Design

As previously mentioned, Tudu will be implemented using the Object Oriented Programming paradigm. Several entities have been established, including Task (and by proxy, Node), List and Manager. Display may also emerge as an entity as the GUI is developed. Our data will be stored in an external file, but while the program is running the data will be kept in linked lists sorted by date.

CLASS DIAGRAM FOR TUDU SOFTWARE



5.1 Manager Class

This class is responsible for managing the flow of data within the software. It will first verify the user’s credentials, then load the lists from the external database, execute the user’s commands and save the changes to the external file. When the program opens and the external data is loaded, the Manager Class will generate three lists: To Do, In Progress and Completed. The three lists will be modified throughout each program session and will be written to the external file as the user logs out or closes the program. Manager Class will also contain several helper functions, such as greeting and farewell messages, and functions to capture and validate user input.

5.2 User Class

This class is used to connect an individual user to their data using their unique ID number. It also stores details about the user including the login name and password needed to verify the user’s credentials.

5.3 List Class

Each Manager Class has three objects of the List Class: To Do, In Progress and Completed. Lists will be implemented as singly linked lists sorted by date. The List Class is responsible for the insertion, deletion, modification and display of objects in the list.

5.4 Task Class

Task Class objects hold the meat of our data. Each object of this class is an item on the to-do list which consists of a field for the description of the task. This class could be easily extended and articulated in greater complexity with more fields for the user to fill in, such as links, location, collaborators, due date, urgency flag, photos and attachments, etc.

5.5 Node Class

Node Class is a derived class of Task Class, containing all attributes of the Task Class with the addition of a “next” pointer to allow it to link up with other nodes in the list. Due to encapsulation, Node Class also contains only two methods used to get and set its sole data member, the “next” pointer.

Ryman