Technical Guide

EPITRELLO

By:

Simpy Surbhi

Rajan

February 9, 2020

Table of Contents

1. Introduction

1.1 Project Background

This project is a command line application to set up a system for managing tasks as well as controlling them. The system comprises of users, lists and tasks. This document is prepared by Simpy Surbhi for their Fundamental Java Project in the Computer Science Master's Program at L'Ecole Pour l'Informatique et les Techniques Avancees (EPITA).

1.2 Project Overview

The application is for effective management of tasks assigned to members. This system is made up of users, lists, and tasks. Each task belongs to a list and can be assigned to a user. The application user create tasks, lists and assign them to members. In addition, for each task priority and estimated time is intended to help us make better decisions about the assignments and tasks.

1.3 Project scope

In Scope :

Creation of users, tasks, lists. Task comprises of user, priority, estimated effort, description, list fields.
Tasks can edited once it is created. It can be assigned to any user.
Users are persisted in h2 DB.
Configuration properties file is added.

Out of Scope :

Tasks are not persisted in memory.
Export option is not available, but if user wants various information those interfaces are present.
No GUI present

1.4 Project Dependencies

In order to run the application, following items are required:

- Install h2 JDBC
- Install Java 8

1.5 Acronyms and Abbreviations

Acronyms	Abbreviations
UML	Unified Modeling Language - a standard way to visualize the design of a system.
JDBC	Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.

2 Business Requirements

S. No.	Requirement Description
1	The system comprises of users, lists and tasks.
2	Each task belongs to a list and can be assigned to a user.
3	Each task has priority and estimated time information.
4	No duplicate users allowed
5	Allow user information to be stored in database
6	Support functions to get the system information like All Task information, All Assigned tasks, All Completed, User Workload, All Unassigned tasks.
7	System should allow to assign task to user, change task status

3 System Specifications

3.1 Functional Requirements

S. No.	Implementation

1.	**Adding users to the system.**
	
	public String addUser(String user) {
	        if(users.contains(user)) {
	            writeToFile("User already exists");
	            return "User already exists";
	        }
	
	        dbManager.addUser(user);
	        users.add(user);
	        writeToFile(SUCCESS);
	        return SUCCESS;
	    }

2.

Adding list to user

```
public String addList(String list) {
            boolean status = lists.add(list);
            if(status) {
                  writeToFile(SUCCESS);
                  return SUCCESS;
            } else {
                  writeToFile("List string already exists");
                  return "List string already exists";
            }
      }
```

3.

Adding new task to the system

```
public String addTask(String list, String name, int estimatedTime, int priority, String
description) {
            if( list == null || list.isEmpty() || name == null || name.isEmpty() ||
estimatedTime <=0 || priority <=0 ) {
                  return "AddTask task failed, reason: entered detail(s) are
invalid!!";
            }

            if(!lists.contains(list)) {
                  writeToFile(LIST_ERROR);
                  return LIST_ERROR;
            }

            Task task = tasks.get(name);
            if(task != null) {
                  writeToFile("Task already exists");
                  return "Task already exists";
            }

            task = new Task();
            task.setDescription(description);
            task.setEstimatedTime(estimatedTime);
            task.setList(list);
            task.setPriority(priority);
            task.setName(name);
            tasks.put(name, task);
            writeToFile(SUCCESS);
            return SUCCESS;
      }
```

4.

Application user is allowed to edit the task

```
public String editTask(String taskName, int estimatedTime, int priority, String
description) {

            if( taskName == null || taskName.isEmpty() || estimatedTime <=0 ||
priority <=0 ) {
```

```
                            return "EditTask task failed, reason: entered detail(s) are
invalid!!";
                  }

                  if(!tasks.containsKey(taskName)) {
                            writeToFile(TASK_ERROR);
                            return TASK_ERROR;
                  }

                  Task task = tasks.get(taskName);
                  task.setDescription(description);
                  task.setEstimatedTime(estimatedTime);
                  task.setPriority(priority);
                  tasks.put(taskName, task);
                  writeToFile(SUCCESS);
                  return SUCCESS;
         }
```

5. **Application user can assign task to user**

```
public String assignTask(String taskName, String user) {
                  if( taskName == null || taskName.isEmpty() || user == null ||
user.isEmpty()) {
                            return "AssignTask task failed, reason: entered detail(s) are
invalid!!";
                  }

                  if(!tasks.containsKey(taskName)) {
                            writeToFile(TASK_ERROR);
                            return TASK_ERROR;
                  }

                  if(!users.contains(user)) {
                            writeToFile(USER_ERROR);
                            return USER_ERROR;
                  }

                  Task task = tasks.get(taskName);
                  task.setUser(user);
                  writeToFile(SUCCESS);
                  return SUCCESS;
         }
```

6. **Delete functions are there to delete system entities**

```
         public String deleteList(String list) {
                  if(list == null || list.isEmpty()) {
                            return "Invalid input";
                  }
                  if(lists.isEmpty() || !lists.contains(list)) {
                            writeToFile(LIST_ERROR);
```

```
            return LIST_ERROR;
        }

        lists.remove(list);
        for(Task task: tasks.values()) {
            if(task.getList().equals(list)) {
                tasks.remove(task.getName());
            }
        }
        writeToFile(SUCCESS);
        return SUCCESS;
    }

    public String deleteTask(String task) {
        if(!tasks.containsKey(task)) {
            return TASK_ERROR;
        }
        tasks.remove(task);
        writeToFile(SUCCESS);
        return SUCCESS;
    }
```

7. **Task related functionality supported**

```
public String completeTask(String taskName) {
        if(!tasks.containsKey(taskName)) {
            writeToFile(TASK_ERROR);
            return TASK_ERROR;
        }
        Task task = tasks.get(taskName);
        task.setStatus(Status.COMPLETE);
        writeToFile(SUCCESS);
        return SUCCESS;
    }

    public String moveTask(String task, String list) {
        if(!lists.contains(list)) {
            writeToFile(LIST_ERROR);
            return LIST_ERROR;
        }

        if(!tasks.containsKey(task)) {
            writeToFile(TASK_ERROR);
            return TASK_ERROR;
        }

        Task tempTask = tasks.get(task);
        tempTask.setList(list);
        tasks.put(task, tempTask);
        writeToFile(SUCCESS);
        return SUCCESS;
}
```

8. **Reporting functionality supported**

```
public String printTotalEstimateTime() {
    if(tasks.isEmpty()) {
        writeToFile("0");
        return "0";
    }

    int effort = 0;
    String user;
    for(Task task: tasks.values()) {
        user = task.getUser();
        if(user == null || user.isEmpty()) {
            continue;
        }
            effort = effort+task.getEstimatedTime();
    }
    writeToFile(""+effort);
    return ""+effort;
}

public String printTotalRemainingTime() {
    if(tasks.isEmpty()) {
        writeToFile("0");
        return "0";
    }

    int effort = 0;
    String user;
    for(Task task: tasks.values()) {
        user = task.getUser();
        if(user == null || user.isEmpty()) {
            continue;
        }

if(task.getStatus().name().equals(Status.NOT_COMPLETE.name())) {
            effort = effort+task.getEstimatedTime();
        }
    }
    writeToFile(""+effort);
    return ""+effort;
}

public String printWorkload() {
    if(users.isEmpty() || tasks.isEmpty()) {
        writeToFile("0");
        return "0";
    }

    int effort = 0;
    String user;
```

```
          for(Task task: tasks.values()) {
                user = task.getUser();
                if(user!=null && !user.isEmpty()) {
                      effort = effort+task.getEstimatedTime();
                }
          }
          writeToFile(""+effort);
          return ""+effort;

    }
```

3.2 Non-functional Requirements

3.2.1 The program shall be able to read a configuration property set from a file on the filesystem which will avoid hardcoded parameters.

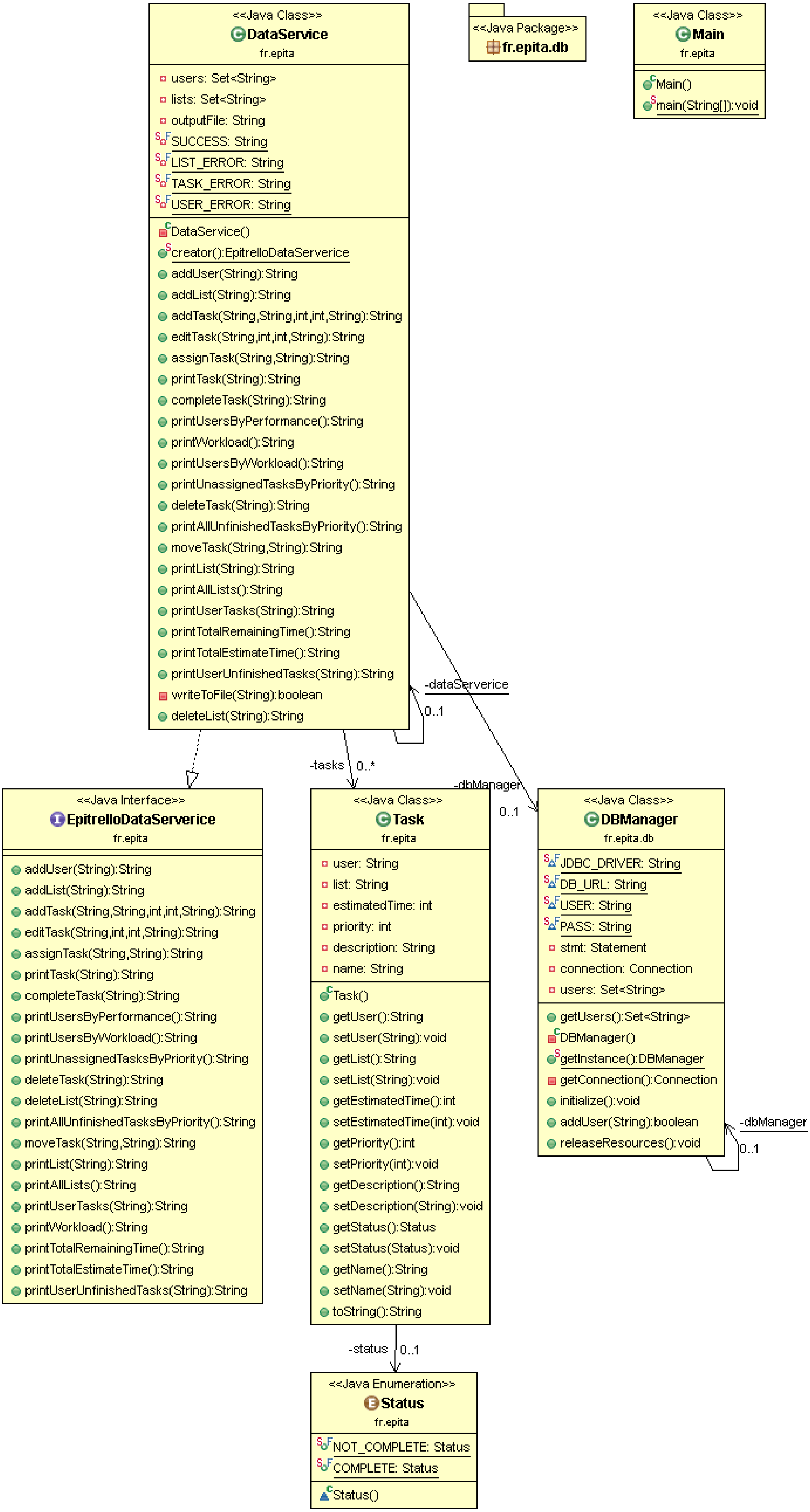
3.2.2 The data is stored in h2 database.

4 Hardware Requirements

S. No.	Hardware	Requirement
1.	Operating System	Compatible with Windows, Mac OS X, Linux
2.	RAM	Minimum required 124MB
3.	Disk Space	Minimum required 124MB
4.	Processor	64-bit, four-core, 2.5 GHz minimum per core

5 Appendix

UML Class Diagram


<<Java Class>>
DataService
fr.epita
users: Set<String>
lists: Set<String>
outputFile: String
SUCCESS: String
LIST_ERROR: String
TASK_ERROR: String
USER_ERROR: String
DataService()
creator():EpitrelloDataServerice
addUser(String):String
addList(String):String
addTask(String,String,int,int,String):String
editTask(String,int,int,String):String
assignTask(String,String):String
printTask(String):String
completeTask(String):String
printUsersByPerformance():String
printWorkload():String
printUsersByWorkload():String
printUnassignedTasksByPriority():String
deleteTask(String):String
printAllUnfinishedTasksByPriority():String
moveTask(String,String):String
printList(String):String
printAllLists():String
printUserTasks(String):String
printTotalRemainingTime():String
printTotalEstimateTime():String
printUserUnfinishedTasks(String):String
writeToFile(String):boolean
deleteList(String):String
<<Java Package>>
fr.epita.db
<<Java Class>>
Main
fr.epita
Main()
main(String[]):void
-dataServerice
0..1
-tasks
0..*
-dbManager
0..1
<<Java Interface>>
EpitrelloDataServerice
fr.epita
addUser(String):String
addList(String):String
addTask(String,String,int,int,String):String
editTask(String,int,int,String):String
assignTask(String,String):String
printTask(String):String
completeTask(String):String
printUsersByPerformance():String
printUsersByWorkload():String
printUnassignedTasksByPriority():String
deleteTask(String):String
deleteList(String):String
printAllUnfinishedTasksByPriority():String
moveTask(String,String):String
printList(String):String
printAllLists():String
printUserTasks(String):String
printWorkload():String
printTotalRemainingTime():String
printTotalEstimateTime():String
printUserUnfinishedTasks(String):String
<<Java Class>>
Task
fr.epita
user: String
list: String
estimatedTime: int
priority: int
description: String
name: String
Task()
getUser():String
setUser(String):void
getList():String
setList(String):void
getEstimatedTime():int
setEstimatedTime(int):void
getPriority():int
setPriority(int):void
getDescription():String
setDescription(String):void
getStatus():Status
setStatus(Status):void
getName():String
setName(String):void
toString():String
<<Java Class>>
DBManager
fr.epita.db
JDBC_DRIVER: String
DB_URL: String
USER: String
PASS: String
stmt: Statement
connection: Connection
users: Set<String>
getUsers():Set<String>
DBManager()
getInstance():DBManager
getConnection():Connection
initialize():void
addUser(String):boolean
releaseResources():void
-dbManager
0..1
-status
0..1
<<Java Enumeration>>
Status
fr.epita
NOT_COMPLETE: Status
COMPLETE: Status
Status()