https://lh6.googleusercontent.com/ZbSIWMo4cow4QNJGA6BveANighDY2_y1CMANa5citrT0ZKaBLxYbxfT-zNV4nV6jZSe7GbYkxIvsv2UQbzEC92oVcW-jMaPgbCVKsddyyd-NH5iW-5ids8pNvEGYhTEQ97jAA1bN

**Computer Science and Engineering**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Taskr**

**Software Design Description**

**Version 1.1**

Document Number: SDD-001

Project Team Number: B12

Project Team Members: Franky Cen Kenan Millet Yatin Kaushal

fc948 kvm237 yk1279

**REVIEW AND APPROVALS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Printed Name and Title** | **Function (Author, Reviewer, Approval)** | **Date** | **Signature** |
| Franky Cen | Author | 10/25/2016 | Franky Cen |
| Yatin Kaushal | Author | 10/25/2016 | Yatin Kaushal |
| Kenan Millet | Author | 10/25/2016 | Kenan Millet |
| Franky Cen | Reviewer | 10/26/2016 | Franky Cen |
| Kenan Millet | Reviewer | 10/26/2016 | Kenan Millet |
| Franky Cen | Approval | 10/26/2016 | Franky Cen |
| Franky Cen | Author | 12/9/2016 | Franky Cen |
| Yatin Kaushal | Author | 12/12/2016 | Yatin Kaushal |
| Kenan Millet | Author | 12/12/2016 | Kenan Millet |
| Franky Cen | Reviewer | 12/15/2016 | Franky Cen |
| Yatin Kaushal | Reviewer | 12/15/2016 | Yatin Kaushal |
| Kenan Millet | Reviewer | 12/15/2016 | Kenan Millet |
| Franky Cen | Approval | 12/15/2016 | Franky Cen |

**REVISION LEVEL**

|  |  |  |
| --- | --- | --- |
| **Date** | **Revision Number** | **Purpose** |
| 10/19/2016 | Version 1.0 | Initial creation of the document |
| 12/9/2016 | Version 1.1 | Finalization of the document |

**TABLE OF CONTENTS**

**1.   Introduction ……………………….………………………………………………………5**

1.1   Purpose ……………………….………………………………………………………5

1.2   Scope ……………………………………………………………………………………5

1.3 Identification …………………………………………………………………………5

1.4 Document Summary …………………………………………………………………5

1.5 System Overview ……………………………………………………………………5

1.6 Document Overview …………………………………………………………………5

**2.   Reference Documents ………………………………………………………………6**

**3.   System Wide Design Decisions ……………………………………………………6**

3.1   Software Component Architectural Design …………………………………………6

3.2   Software Architectural General Description …………………………………………6

3.3 Software Item Components …………………………………………………………6

3.4 Component Interface Identification ……………………………………………………6

3.5 Software Component Concept of Execution …………………………………………6

**4. Software Item Detailed Design ……………………………………………………6**

4.1   Structure …………………………………………………………………………6

4.1.1 Software Unit Detailed Design …………………………………………………6

4.2   Static Relationship of Software Unit …………………………………………………6

4.2.1 Run-Time Object Instances ……………………………………………………6

4.3 Behavior ……………………………………………………………………………7

4.3.1 Interaction Diagrams …………………………………………………………7

4.3.2 Collaboration Diagrams ………………………………………………………7

4.3.3 Activity Diagrams ……………………………………………………………8

4.4   Concept of Execution ………………………………………………………………8

4.5   Interface Design ……………………………………………………………………8

4.5.1 Interface Identification and Diagrams …………………………………………8

4.5.2 Unique Identifier of Interface …………………………………………………8

**5.   Implementation Architecture …………………………………………………………8**

5.1   All Active and Passive Classes Assigned to Components …………………………8

5.2   Diagrams of Physical Packaging of Logical Components …………………………8

**6.   Deployment Architecture …………………………………………………………8**

6.1   Physical Deployment Architecture Diagram ………………………………………8

**7.   Pseudocode …………………………………………………………………………8-9**

**8.   Dictionaries …………………………………………………………………………9**

**9. Software Item Computer Resource Utilization …………………………………9**

**10. Requirements Traceability ……………………………………………………………9**

10.1   Software Component Level Requirements Traceability ……………………………9

**11. System Design Testing ……………………………………………………………10**

**12. Rationale …………………………………………………………………………………10**

**13.   Appendix ………………………………………………………………………………11**

13.1   Schedule Tracking …………………………………………………………………11

13.2   Defect Tracking ………………………………………………………………………12

13.3.1 Component Architecture Diagram ………………………………………………14

13.4.1.1 Class Diagrams …………………………………………………………………15

13.4.3.1 Sequence Diagrams ……………………………………………………………16

13.4.3.3 Activity Diagram ………………………………………………………………17

13.6.1 Deployment Architecture Diagram ………………………………………………18

13.7 Dictionaries ……………………………………………………………………19-23

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the contents of the design of the system. The document is used to communicate overall quantitative and qualitative system characteristics to operations management, technical support, training, and operators.

**1.2 Scope**

This product is expected to help users organize their schedules. Users will interact with the product, which can help create a schedule that most benefits the user. The information for the tasks and schedules are saved both locally and externally, on a server. This product will not force users to follow their schedule, or penalize them for not doing so. However, if users do not follow their schedule, this product will adjust its suggestions accordingly. This product can be for personal, commercial, or business use.

**1.3 Identification**

Taskr Software Design Description v1.0

**1.4 Document Summary**

This document is meant to outline the design for Taskr in a software perspective. It does so by describing the classes and methods that will be implemented. This document includes diagrams that illustrate how certain components of the product will work in relation with each other.

**1.5 System Overview**

The users will interact with the system by making requests and receiving formatted data in the form of schedules and tasks. The system will interact with the database, which will store the data relevant to each user in an unformatted, but organized manner.

**1.6 Document Overview**

Document:

Title Page (Standard Cover)

Review/Approval Signatures

Table of Revisions (revision number, date, purpose)

Table of Contents

Figures

Schedule and Fault tracking

Class, Sequence, Activity, Physical Deployment Diagrams

This project follows the Iterative and Incrementation Life Cycle Model.

All Diagrams, Use Cases, and Dictionaries follow the Object Oriented UML standard.

**2. Reference Documents**

Taskr Project Proposal v1.0 02/17/2016

Objectives

Rationale

Bounds

Taskr Software Requirements Specification v1.1 03/23/2016

Business Requirements

Functional and Nonfunctional Requirements

System Test Plan Requirements

Taskr System Analysis Specification v1.0 04/11/2016

Class Diagrams

Context Diagrams

Architecture Diagrams

Dictionaries

Event Diagrams

Taskr System Project Management Plan v1.1 09/27/2016

Various Test and Management Plans

**3.   System Wide Design Decisions**

**3.1   Software Component Architectural Design**

See Appendix 13.3.1 for the Component Architecture Diagram

**3.2   Software Architectural General Description**

This product can be used in a personal, school, commercial, or business environment. This application aims to increase efficiency by providing users with a schedule that allows them to use their time optimally. Users can create tasks and schedules that the application can arrange so that the user is most benefited. The user interacts with the program through a smartphone’s touch screen. The information taken from the user is used to generate and display a schedule that fits the user’s preferences.

**3.3 Software Item Components**

Software Components:

* Core Module
* Interface Module
* Database Module
* Brain Module

Component Descriptions are located in the RAS (Section 6.4.2)

**3.4 Component Interface Identification**

Component Interfaces:

* Core Module to Database Module
  + The Core Module can store and fetch information from the database
* Core Module to Brain Module
  + The Core Module uses the Brain Module to compute and return information
  + Core Module to Interface Module
    - The Interface Module requests the Core Module to start events
    - The Core Module returns information based on the requested event
  + Interface Module to User
    - The Interface Module displays information to the user using the smartphone screen
    - The User requests the start of events by giving input to the touch-screen

**3.5 Software Component Concept of Execution**

An event is started when the user interacts with the smartphone. The Interface Module takes this input and starts events based on the user’s selections. The Core Module handles this request. If calculations are required, the Brain Module is used to handle calculations. The data from the database needs to be fetched, the Core Module will interact with the Database Module, which takes information from the server. The information is passed along the Components to be displayed to the user.

**4. Software Item Detailed Design**

**4.1   Structure**

**4.1.1 Software Unit Detailed Design**

See Appendix 13.4.1.1 for Class Diagrams

**4.2   Static Relationship of Software Unit**

**4.2.1 Run-Time Object Instances**

When the User requests for an event to happen, the Application runs a function that calls the Server. The Server runs a corresponding function to call the Database. The Database runs a function to complete the event that the user requested. If information needs to be returned, the Database returns the information to the Server, which returns the information to the Application, which displays the information to the User.

Example: The User requests to create a new schedule.

The Application calls the method createSchedule(), which calls the Server.

The Server calls *its* method createSchedule(), which calls the Database.

The Database runs createSchedule().

A new Schedule object is created and saved within the Database.

**4.3 Behavior**

**4.3.1 Interaction Diagrams**

See Appendix 13.4.3.1 for Sequence (Interaction) Diagrams

**4.3.2 Collaboration Diagrams**

Collaboration Diagrams are to be delivered at a later date.

**4.3.3 Activity Diagrams**

See Appendix 13.4.3.3 for Activity Diagrams

**4.4   Concept of Execution**

The system is launched by the user when the phone application is started. The application will retrieve information from the database. When the user starts an event (events are listed in the SAS), the application runs the corresponding function. The database within the server is used to store information. The application can read and write information in the database.

**4.5   Interface Design**

**4.5.1 Interface Identification and Diagrams**

Section 3.4 identifies and describes all system interfaces.

Appendix 13.3.1 Component Architecture Diagram shows the interactions between components, and shows the interfaces between them. The interfaces are the connections between Modules (components).

**4.5.2 Unique Identifier of Interface**

**5.   Implementation Architecture**

**5.1   All Active and Passive Classes Assigned to Components**

**5.2   Diagrams of Physical Packaging of Logical Components**

Section 5 to be delivered at a later date.

**6.   Deployment Architecture**

**6.1   Physical Deployment Architecture Diagram**

See Appendix 13.6.1 for Deployment Architecture Diagram

**7. Pseudocode**

// Schedule function::

bool isTimeslotFree(Date startTime, Date endTime) {

if ( startTime < this.start OR endTime > this.end )

return false;

for each Task task in taskList:

if ( startTime < task.getEndDate()

AND endTime > task.getStartDate() )

return false;

else if ( task.getStartDate() < endTime

AND task.getEndDate() > startTime )

return false;

else if ( task.getStartDate() == startDate

OR task.getEndDate() == endDate )

return false;

return true;

}

// Brain function::

List <Task> splitTasks( List <Task> tasks, float timeInterval ) {

List <Task> newTaskList

sort tasks by the Manual property

for each Task in tasks {

if the task has the Manual property: add it to the newTaskList.

else {

calculate how many parts to split the Task into

create new tasks that are smaller Parts of the original Task

add each Part to the newTaskList

}

}

return newTaskList

}

// Brain function::

Schedule generateSchedule(Date begin, Date end, List <Task> tasks, float timeInterval) {

Schedule sched = new Schedule( “Taskr Generated Schedule”, begin, end)

List <Task> splits = splitTasks(tasks, timeInterval)

add each Task in splits to the sched

// the added Tasks will not have a set startDate or endDate yet

sort the sched’s taskList by priority

for each Task in sched’s taskList:

set the Task’s startDate and endDate

return sched

}

**8.   Dictionaries**

See Appendix 13.7 for Dictionaries

**9. Software Item Computer Resource Utilization**

The product will be an application that is used on Android smartphones.

The products requirements are listed below:

* Smartphone that has enough space to hold the program.
* Connection to the server
* Access to the database
* Database must have enough space to hold all stored information

**10. Requirements Traceability**

A traceability matrix will be used to ensure requirement traceability. The traceability matrix is a document in the form of a table that can be used to check whether or not the current requirements are being met. New requirements are added to the traceability matrix. The relationships of each succeeding artifact to their source document will be recorded so that there is both forward and backward traceability. When a requirement is changed in a source document, changes that need to be done in documents can be determined using the traceability matrix. Changes will be requested, reviewed, then approved before being applied.

**11.   System Design Testing**

Once completed and approved, this document will be given to the Software Quality Group, who will develop the test plan and a set of test scenarios based on the Use Cases, execute the tests and report any defects. Each feature of the product will be tested against several scenarios. This testing will ensure that the product functions correctly. The testing will be conducted using an SQL server.

**12. Rationale**

Time management is a problem that many in first world countries face. The common saying goes that there are only twenty-four hours in the day, but if used efficiently, twenty-four hours can be a very long time.

We believe that the solution to efficient use of time is through the proper mindset. Unfortunately, due to entertainment, stress, friends, family, and many other important but deviating tasks, such a mindset can be very difficult to adopt and equally difficult to maintain. In order to help the population adapt to the mindset of the successful, we will develop an application that acts as a scheduler that can be used to help ease people into managing their time efficiently.

**13.   Appendices**

**13.1   Schedule Tracking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact / Deliverable** | **Individual** | **Estimated** | **Actual** | **Difference** |
| SRS - Initial | Franky  Kenan  Yatin | 14 hours  12 hours  15 hours | 8 hours  14 hours  7 hours | 6 hours  2 hours  8 hours |
| **Total** | **41 hours** | **29 hours** | **12 hours** |
| SRS - Final | Franky  Kenan  Yatin | 15 hours  20 hours  15 hours | 9 hours  6.5 hours  5 hours | 6 hours  12.5 hours  10 hours |
| **Total** | **50 hours** | **20.5 hours** | **29.5 hours** |
| SPMP - Initial | Franky  Kenan  Yatin | 13 hours  18 hours  10 hours | 16.5 hours  12 hours  7 hours | 3.5 hours  6 hours  3 hours |
| **Total** | **41 hours** | **35.5 hours** | **5.5 hours** |
| SAS | Franky  Kenan  Yatin | 15 hours  12 hours  15 hours | 13 hours  7.5 hours  8 hours | 2 hours  4.5 hours  7 hours |
| **Total** | **42 hours** | **28.5 hours** | **13.5 hours** |
| SPMP - Final | Franky  Kenan  Yatin | 8 hours  3 hours  2 hours | 4 hours  2 hours  1 hours | 4 hours  1 hour  1 hour |
| **Total** | **13 hours** | **7 hours** | **6 hours** |
| RAS | Franky  Kenan  Yatin | 10 hours  4 hours  3 hours | 3.5 hours  4 hours  1.5 hours | 6.5 hours  0 hours  1.5 hours |
| **Total** | **17 hours** | **8 hours** | **9 hours** |
| SDD - Initial | Franky  Kenan  Yatin | 15 hours  6 hours  5 hours | 4.5 hours  2.5 hours  2 hours | 10.5 hours  3.5 hours  3 hours |
| **Total** | **26 hours** | **9 hours** | **18 hours** |
| SDD - Final | Franky  Kenan  Yatin | 6 hours  5 hours  5 hours | 6.5 hours  1.5 hours  2 hours | 0.5 hours  3.5 hours  3 hours |
| **Total** | **16 hours** | **10 hours** | **7 hours** |

**13.2   Defect Tracking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Artifact / Deliverable** | **Individual** | **Estimated** | **Actual** | **Difference** |
| SRS - Initial | Franky  Kenan  Yatin | 20 faults  10 faults  10 faults | 10 faults  6 faults  3 faults | 10 faults  4 faults  7 faults |
| **Total** | **40 faults** | **19 faults** | **21 faults** |
| SRS - Final | Franky  Kenan  Yatin | 20 faults  20 faults  15 faults | 24 faults  2 faults  3 faults | 4 faults  18 faults  12 faults |
| **Total** | **55 faults** | **29 faults** | **26 faults** |
| SPMP - Initial | Franky  Kenan  Yatin | 20 faults  20 faults  20 faults | 16 faults  17 faults  4 faults | 4 faults  3 faults  16 faults |
| **Total** | **60 faults** | **37 faults** | **23 faults** |
| SAS | Franky  Kenan  Yatin | 15 faults  20 faults  18 faults | 13 faults  12 faults  15 faults | 2 faults  8 faults  3 faults |
| **Total** | **53 faults** | **40 faults** | **13 faults** |
| SPMP - Final | Franky  Kenan  Yatin | 15 faults  10 faults  10 faults | 10 faults  4 faults  3 faults | 5 faults  6 faults  7 faults |
| **Total** | **35 faults** | **15 faults** | **18 faults** |
| RAS | Franky  Kenan  Yatin | 15 faults  20 faults  5 faults | 14 faults  12 faults  1 fault | 1 fault  8 faults  4 faults |
| **Total** | **40 faults** | **27 faults** | **13 faults** |
| SDD - Initial | Franky  Kenan  Yatin | 25 faults  20 faults  15 faults | 11 faults  5 faults  9 faults | 14 faults  15 faults  6 faults |
| **Total** | **60 faults** | **25 faults** | **35 faults** |
| SDD - Final | Franky  Kenan  Yatin | 10 faults  8 faults  12 faults | 7 faults  2 faults  2 faults | 3 faults  6 faults  10 faults |
| **Total** | **30 faults** | **11 faults** | **19 faults** |

**Cumulative Schedule Tracking**

|  |  |  |  |
| --- | --- | --- | --- |
| **Individual** | **Estimated** | **Actual** | **Difference** |
| Franky  Kenan  Yatin | 96 hours  80 hours  70 hours | 65 hours  50 hours  33.5 hours | 31 hours  30 hours  36.5 hours |
| **Total** | **246 hours** | **148.5 hours** | **97.5 hours** |

**Cumulative Defect Tracking**

|  |  |  |  |
| --- | --- | --- | --- |
| **Individual** | **Estimated** | **Actual** | **Difference** |
| Franky  Kenan  Yatin | 140 faults  128 faults  105 faults | 105 faults  60 faults  40 faults | 35 faults  68 faults  65 faults |
| **Total** | **373 faults** | **205 faults** | **168 faults** |

**13.3.1 Component Architecture Diagram**

Component Architecture Diagram.png

**13.4.1.1 Class Diagrams**



\*Omitted simple getters and setters for each Attribute.

\*\* Database class includes getters and setters for each stored Task and Schedule in a List.

The Relationships are detailed in the Dictionaries (section 13.7).

**13.4.3.1 Sequence Diagram**



**13.4.3.3 Activity Diagram**



**13.6.1 Deployment Architecture Diagram**



**13.7 Dictionaries**

**Classes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Methods** | **Attributes** |
| Task | User created Tasks / Events | \* | String name  Date startDate  Date endDate  float start  float duration  float desirability  Date urgency  float importance  bool manual  float priority  float completion  String notes |
| Schedule | User created Schedules | \*  boolean isTimeslotFree(Date start, Date end)  void addTask(Task task)  void removeTask(int taskNum)  void editTask(int taskNum) | String name  List <Task> taskList  Date start  Date end  String notes |
| Database | Storage of all classes and data. | \*\*  void createTask()  void addTask()  void removeTask()  void editTask()  List <Task> getManualTasks()  List <Task> getAutoTasks()  List <Task> getSortedTaskListPriority()  void createSchedule()  void addSchedule(Schedule sched)  void editSchedule(int scheduleNum)  void removeSchedule(int scheduleNum)  void addTaskToSchedule(int taskNum, int scheduleNum)  void removeTaskToSchedule(int taskNum, int scheduleNum) | List <Task> taskList  List <Schedule>  scheduleList |
| Brain | Calculation / Algorithms | Schedule generateSchedule(Date begin, Date end,  List <Task> tasks)  List <Task> splitTasks(List <Task> taskList, float timeInterval) | float desWeight  float impWeight  float durWeight  float urgWeight |
| Interface | Handles User Input.  Displays information to the User. | void createTask()  void editTask()  void removeTask()  void createSchedule()  void editSchedule()  void removeSchedule()  void displayTasks()  void displaySchedules()  void addTaskToSchedule()  void removeTaskFromSchedule()  void generateSchedule() |  |

\*Omitted simple getters and setters for each Attribute.

\*\* Database class includes getters and setters for each stored Task and Schedule in a List.

**Methods\***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Class** | **Arguments** |
| bool isTimeslotFree | Tests to see if the Schedule is open between certain times. | Schedule | Date startDate  Date endDate |
| void addTask | Adds a Task to the Schedule | Schedule | Task newTask |
| void editTask | Edits a Task in the Schedule | Schedule  Database | int taskNum |
| void removeTask | Removes a Task from the Schedule | Schedule | int taskNum |
| void createTask | Creates a Task and adds it to the Database | Database | \*\* |
| void addTask | Adds a Task to the Database | Database | Task myTask |
| void deleteTask | Removes a Task from the Database. | Database | int taskId |
| List <Task> getManualTasks | Gets a list of all Tasks with the manual property. | Database |  |
| List <Task> getAutoTasks | Gets a list of all non-manual Tasks. | Database |  |
| List <Task>  getSortedTaskListPriority() | Gets a sorted list of all Tasks.  The list is sorted by ascending Task priority | Database |  |
| void createSchedule | Creates a Schedule and adds it to the Database | Database | \*\* |
| void addSchedule | Adds a Schedule to the Database. | Database | Schedule mySched |
| void editSchedule | Edits a Schedule in the Database | Database | int scheduleNum |
| void removeSchedule | Removes a Schedule from the Database. | Database | int scheduleNum |
| void addTaskToSchedule | Adds a Task to a Schedule | Database | int taskNum  int scheduleNum |
| void removeTaskFromSchedule | Removes a Task from a Schedule | Database | int taskNum  int scheduleNum |
| Schedule generateSchedule | Generates a schedule based on a list of given Tasks, and a set time period. | Brain | Date startDate  Date endDate  List <Task> tasks |
| void createTask | Gets User input to create a Task. | Interface | \*\* |
| void editTask | Edits a Task based on User input. | Interface | int taskNum |
| void removeTask | The User requests to remove a Task. | Interface |  |
| void createSchedule | Gets User input to create a Schedule. | Interface | \*\* |
| void editSchedule | Edits a Schedule based on User input. | Interface | int scheduleNum |
| void removeSchedule | The User requests to remove a Schedule. | Interface | int scheduleNum |
| void displayTasks | The User requests to view all Tasks. | Interface |  |
| void displaySchedules | The User requests to view all Schedules. | Interface |  |
| void addTaskToSchedule | Adds a User selected Task to a Schedule. | Interface | int taskNum  int scheduleNum |
| void removeTaskFromSchedule | Removes a User selected Task from a Schedule. | Interface | int taskNum  int scheduleNum |
| void generateSchedule | A Schedule is generated based on User selected Tasks over a set time period. | Interface | Date startDate  Date endDate  List <Task> tasks |

\*Omitted simple getters and setters for each Attribute.

\*The Database holds a List of Tasks and Schedules, and is able to call each Task and Schedule method.

\*\*For createTask() and createSchedule(), the arguments are each Attribute of Tasks and Schedules, respectively.

**Attributes**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Simple/**  **Complex** | **Description** | **Class** | **Type** | **Attributes** | **R/W** |
| name | Simple | Name of the Task or Schedule. | Task  Schedule | String |  | R/W |
| startDate | Complex | The starting date of the Task or Schedule. | Task  Schedule | Date | time, date, year  day of week | R/W |
| endDate | Complex | The ending date of the Task or Schedule | Task  Schedule | Date | time, date, year  day of week | R/W |
| startTime | Simple | The starting time of the Task (ex: 4 PM). | Task | float |  | R/W |
| duration | Simple | The duration of the Task (ex: 2 hours). | Task | float |  | R/W |
| desirability | Simple | The desirability of the Task (Scale 1-10). | Task | float |  | R/W |
| urgency | Complex | The deadline of the Task.  (On a Scale of 1 to 10). | Task | Date | time, date, year  day of week | R/W |
| importance | Simple | The importance of the Task (Scale 1-10). | Task | float |  | R/W |
| manual | Simple | Whether the Task has a fixed timeslot  (ex: A meeting on 4/12, 3:00 PM). | Task | bool |  | R/W |
| priority | Simple | The priority of the Task.  This is calculated by the system.  Never seen by the User. | Task | float |  |  |
| completion | Simple | The percentage the Task is finished. | Task | float |  | R/W |
| notes | Simple | User notes for their Tasks or Schedules. | Task  Schedule | String |  | R/W |
| taskList | Simple | A Schedule has a List of Tasks in it.  The Database has a List of all Tasks. | Schedule  Database | List |  | R/W |
| timeslots | Simple | A time chart in the form of a 2-dimensional array.  timeslots[ 1 ][ 1200 ] refers to day #1, 12 PM.  If timeslots[ day ][ time ] = 0,  the schedule is free at that day and time.  If timeslots[ day ][ time ] = 1,  the schedule is not free at the day and time. | Schedule | int[][] |  | R/W |
| schedList | Simple | A List of stored Schedules. | Database | List |  | R/W |
| desWeight | Simple | System variables.  Used for Schedule generation.  Modified only by the System.  Not seen by the user. | Brain | float |  |  |
| impWeight | Simple | Brain | float |  |  |
| durWeight | Simple | Brain | float |  |  |
| urgWeight | Simple | Brain | float |  |  |

**Relationships**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Description** | **From Class** | **To Class** | **Optional/**  **Mandatory** | **Cardinality** |
| User Interaction | The User interacts with the Interface,  which interacts with the Database. | Interface | Database | Mandatory | One-to-One |
| Calculation | The Database calls for the Brain to  generate Schedules. | Database | Brain | Mandatory | One-to-One |
| Task Storage | The Database stores a list of all Tasks | Database | Task | Mandatory | One-to-Many |
| Schedule Storage | The Database stores a list of all Schedules | Database | Schedule | Mandatory | One-to-Many |
| Schedule-Task  Storage | The Schedule stores a list of Tasks | Schedule | Task | Mandatory | One-to-Many |