**Project Management System**

**First Sprint**

*All requirements and specifications gathered from:*

*“Project Management System Requirements v3a” document*

**Designed For**

**Professor Lawrence Stein**

**Project Management Institute**

**Designed by**

**Vartan Kazaryan** – Database & Middleware Design, Deliverables Design

**Milaud** – GUI Design

**Luke** – Issues and Tasks Design

**Kian** – Decisions Design

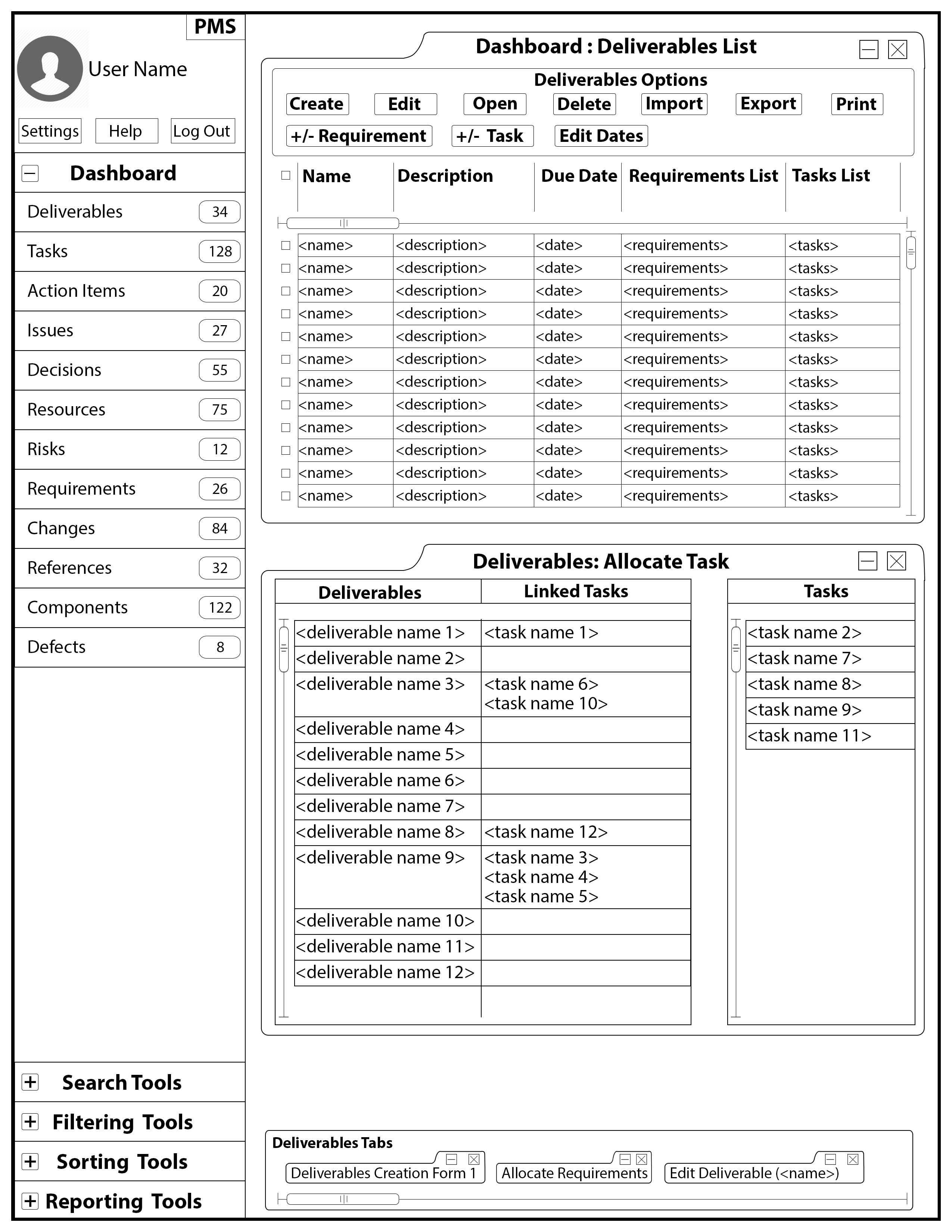
**Iman** – Resources and Action Items Design

**Table of Contents**

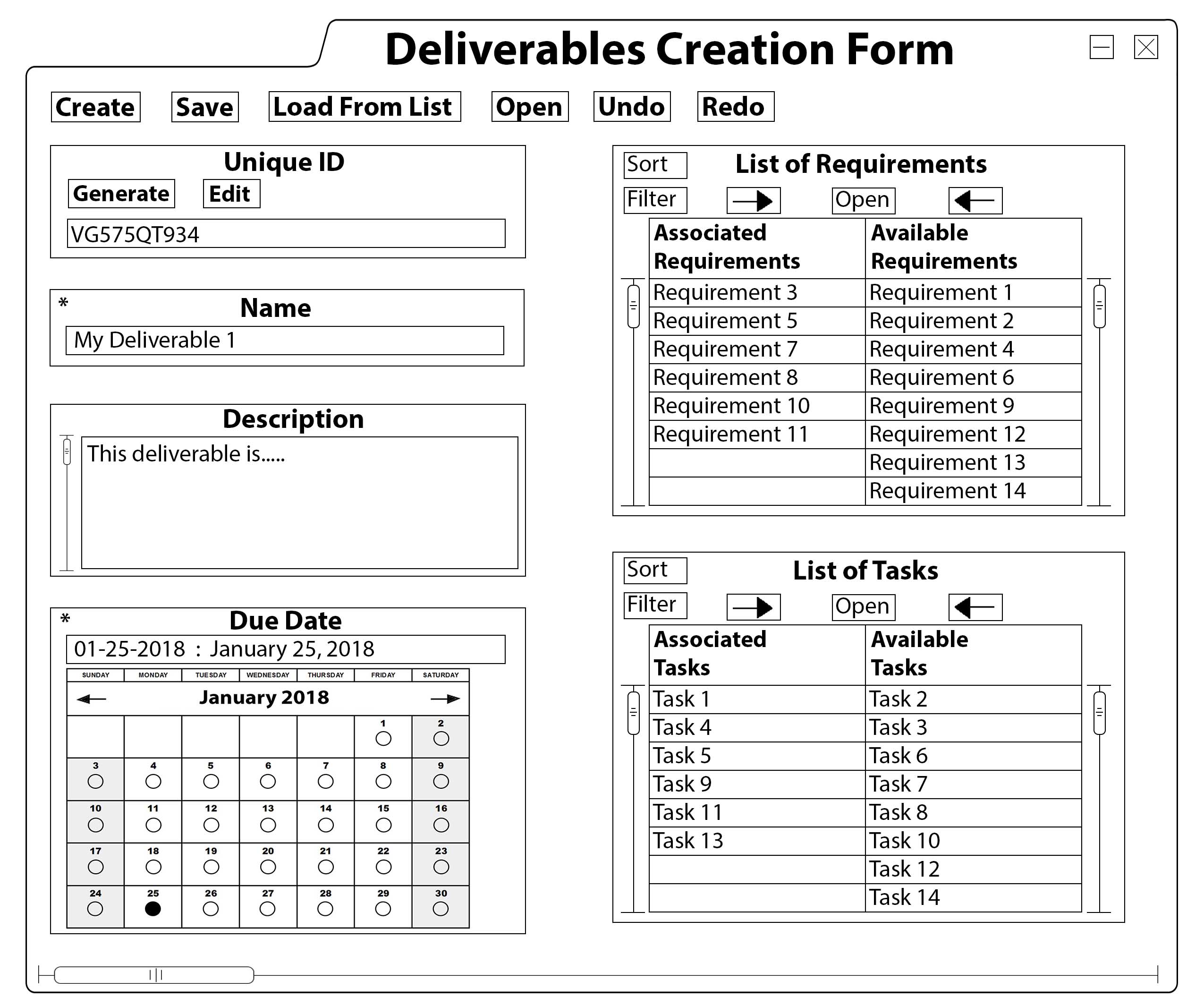
1. [**Storyboard**](#Storyboard)
2. [**High Level Design**](#High_Level_Design)
   1. **Class Diagrams**
      1. Deliverables
      2. Requirements (has nested classes)
      3. Tasks
      4. Resources (has nested classes)
      5. Issues
      6. Action Items
      7. Decisions
   2. [**Use Case Diagrams**](#Use_Case_Diagrams)
      1. Deliverables
      2. Requirements
      3. Tasks
      4. Resources
      5. Issues
      6. Action Items
      7. Decisions
   3. [**Three-Tier Diagram**](#Three_Tier_Diagram)
3. **GUI Prototype**
4. [**Detailed Design**](#Detailed_Design)
   1. [**Entity Relationship Diagrams**](#Entity_Realtionship_Diagrams)
      1. Deliverables Entity Table
      2. Requirements Entity Table
         1. Page References Entity Table
      3. Tasks Entity Table
         1. Summary Tasks Entity Table
         2. Milestones Entity Table
         3. Composite Tasks Entity Table
      4. Resources Entity Table
         1. Skills Entity Table
         2. Days Available Entity Table
      5. Issues Entity Table
      6. Action Items Entity Table
      7. Decisions Entity Table
         1. MeetingNotes Entity Table
         2. ReferenceDocuments Table
      8. Decision-Task Bridge Table
      9. Issue-Task Bridge Table
      10. DropDownLists Component Table
      11. Full System Relationship Diagram
   2. [**Data Flow Diagram**](#Data_Flow_Diagram)
5. [**Test Cases**](#Test_Cases)
6. **Requirements Cross Reference Matrix**
7. **Code Summary**
8. **Storyboard**

**Overview:** The following sections for the storyboard are all taken from early mock-up designs. Many aspects have been changed or updated to reflect newly found implied/derived requirements gathered from the clients’ requirement specifications document. There are three main GUI components for all functional requirements. The main table which lists all items that belong to that category in a tabular form. The Creation form which allows the user to create an item by sending in various inputs to the GUI fields. Then finally there is the open GUI component which displays all of the data that belongs to a single item, this includes components that are lists of other items. All of the other functional requirement GUI components follow the template set by the three figures posted below.

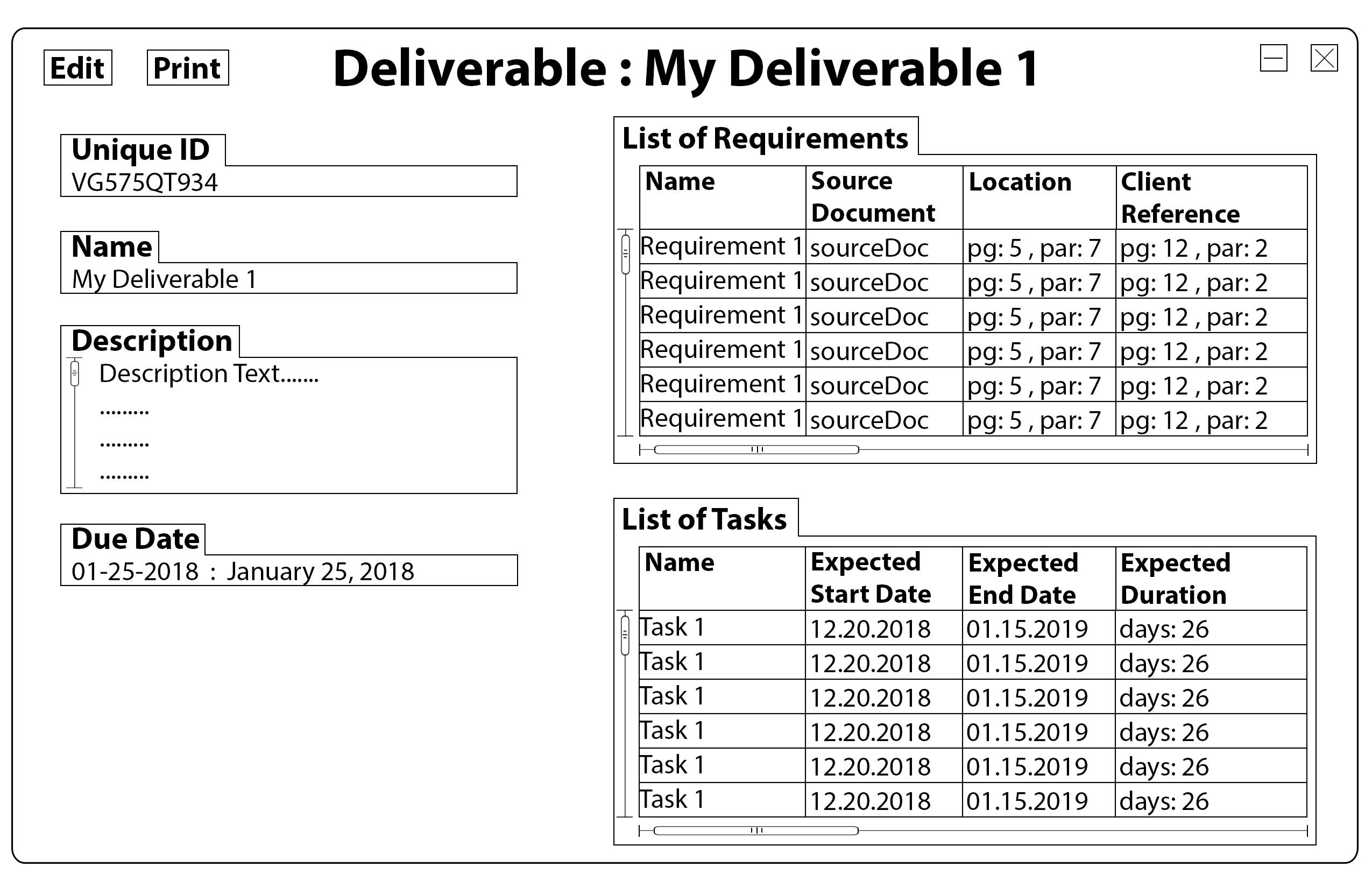
*Figure 1.1: Deliverables Table*



*Figure 1.2: Deliverable Creation Form*



*Figure 1.3: Deliverable Item Display*



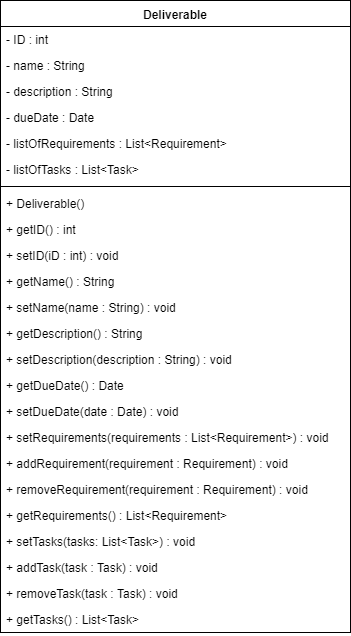
1. **High Level Design**

**2.1.** **Class Diagrams:** Class diagrams show the structure of the objects which represent the data of all functional requirements listed. It should be noted that while these class diagrams do store data, this is only the temporary data which gets created when a GUI component requests information. An example would be a list of deliverables being created, then populated with deliverable data whenever the deliverable list GUI is opened. All of the actual data is stored in a persistent database system called Mysql. More information about the system will be explained in greater detail later in this document.

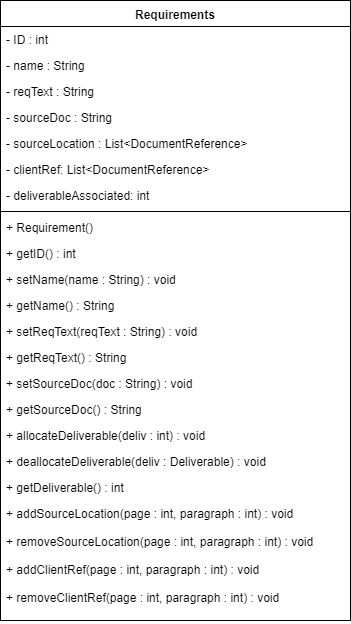
All functional requirements have one attribute that function the same way. The **ID** attribute is a 10-digit numeric identifier which gets auto-generated when a new functional item is created. This identifier cannot be changed or modified in any way (internally or by the user). Item creation will increment the ID of that category by 1, removing an item in a category will not change the ID increment values.

There is also another class which many functional types use for their attributes called **DropDownList**. This class is a helper component class that stores various options for a drop down list. The information for these classes is pulled directly from the database upon request from the GUI. When the object is first created, the constructor will pass in default values for the options that are hard coded in. These options belong specifically to the individual component in the functional type class.

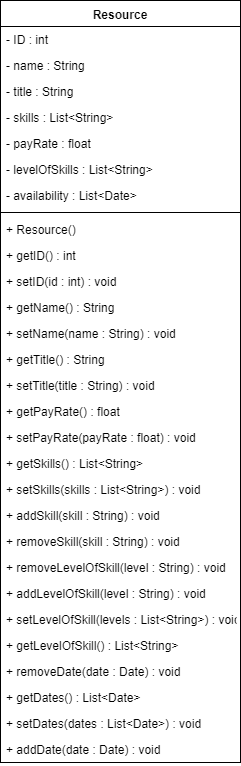
* + 1. **Deliverables:** This class includes all of the information necessary for a Project Manager to define a tangible or intangible good or service that is to be delivered to the customer. The deliverable class contains methods to manage all attributes; **name, description, dueDate, listOfRequirements, and listOfTasks**. All of these class members correlate directly to the attribute requirements specified by the client in their attributes table. The constructor of the deliverable will only initialize all attributes to default empty values to ensure the program doesn’t experience a hard crash. The handling methods for each attribute are explained and listed below:
* **Name:** This indicates the name of the deliverable. The public **get()** method will return the current value of the name variable. The public **set(name : String)** method will take in a string parameter and set the value of the name attribute to whatever the parameter is. Duplicate names are allowed, but are not recommended. Names must be less than 50 characters long, the validity of the attribute is checked in the GUI component during runtime. A name is required and must not be empty for a deliverable to be created or edited.
* **Description:** This is a short description of the deliverable. The public **get()** method will return the current value of the description variable. The public **set(description : String)** method will take in a string parameter and set the value of the description attribute to whatever the parameter is. Descriptions must be less than 500 characters long, the validity of the attribute is checked in the GUI component during runtime. A description is not needed and can be left empty.
* **DueDate:** This indicates when the deliverable is due and notes when it must be turned in to the client. Deliverables must have a due date and are typically determined by the clients’ requirements. The public **get()** method returns a built in Java Date object. This Date object holds values for the year, month, day, as well as a timestamp (though the timestamp is not used). The public **set(date : Date)** method takes in a Date parameter and sets the dueDate attribute to it. The values of the date have to be past the local date.
* **ListofRequirements:** This is a list of all the requirements which are associated with this deliverable. The list indicates what is required for the deliverable to be completed. A Java object type of List is used to store the requirements in the deliverable. The public **setRequirements(requirements : List<Requirement>)** method takes in a list of requirements as a parameter and sets the attribute to that. The **addRequirement(requirement : Requirement)** method takes in a specific requirement as a parameter and adds it to the attributes current list of requirements. The **removeRequirement(requirement : Requirement)** method takes in a specific requirement as a parameter and removes it from the attributes current list of requirements. The **getRequirements()** method returns the list of requirements.
* **ListofTasks:** This is a list of all the tasks which are associated with this deliverable. The list indicates all the tasks that must be completed before the deliverable is completed and ready for deployment. A Java object type of List is used to store the tasks in the deliverable. The public **setTasks(tasks : List<Task>)** method takes in a list of tasks as a parameter and sets the attribute to that. The **addTask(task : Task)** method takes in a specific task as a parameter and adds it to the attributes current list of tasks. The **removeTask(task : Task)** method takes in a specific task as a parameter and removes it from the attributes current list of tasks. The **getTasks()** method returns the list of tasks.



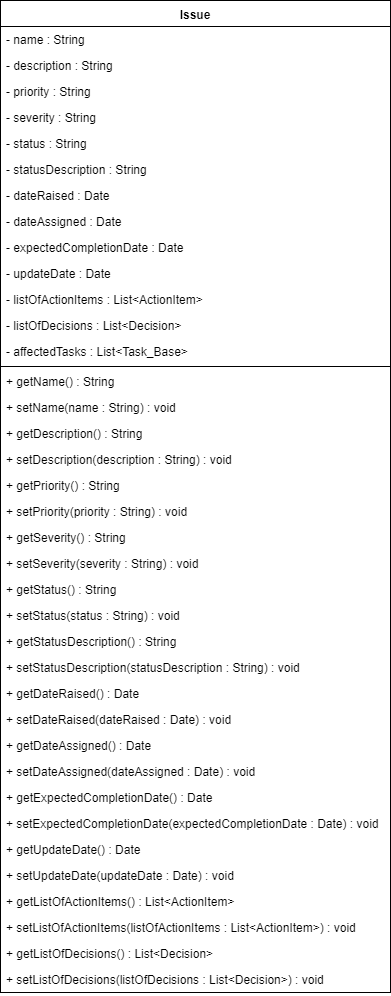
* + 1. **Requirements**



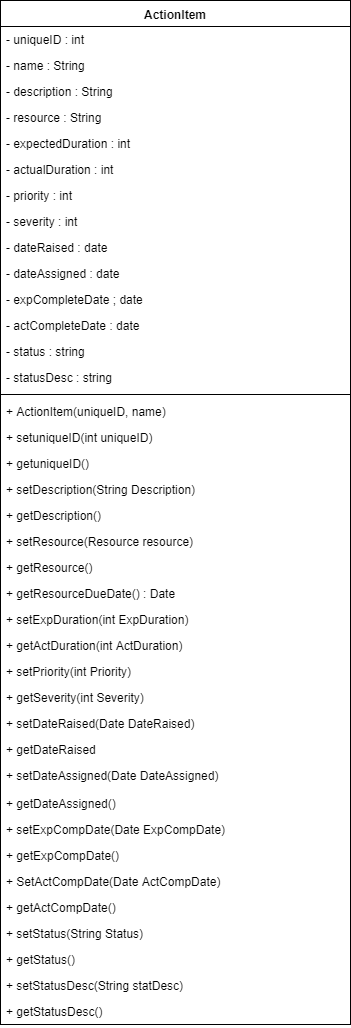
* + 1. **Tasks**
    2. **Resources**



* + 1. **Issues:**



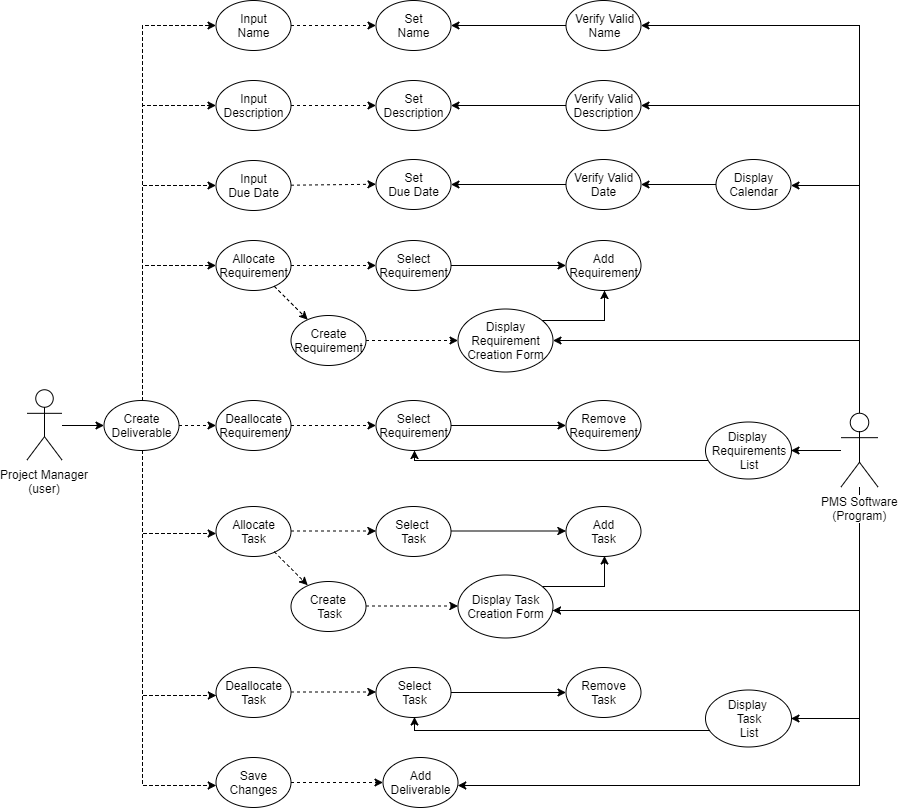
* + 1. **Action Items:**



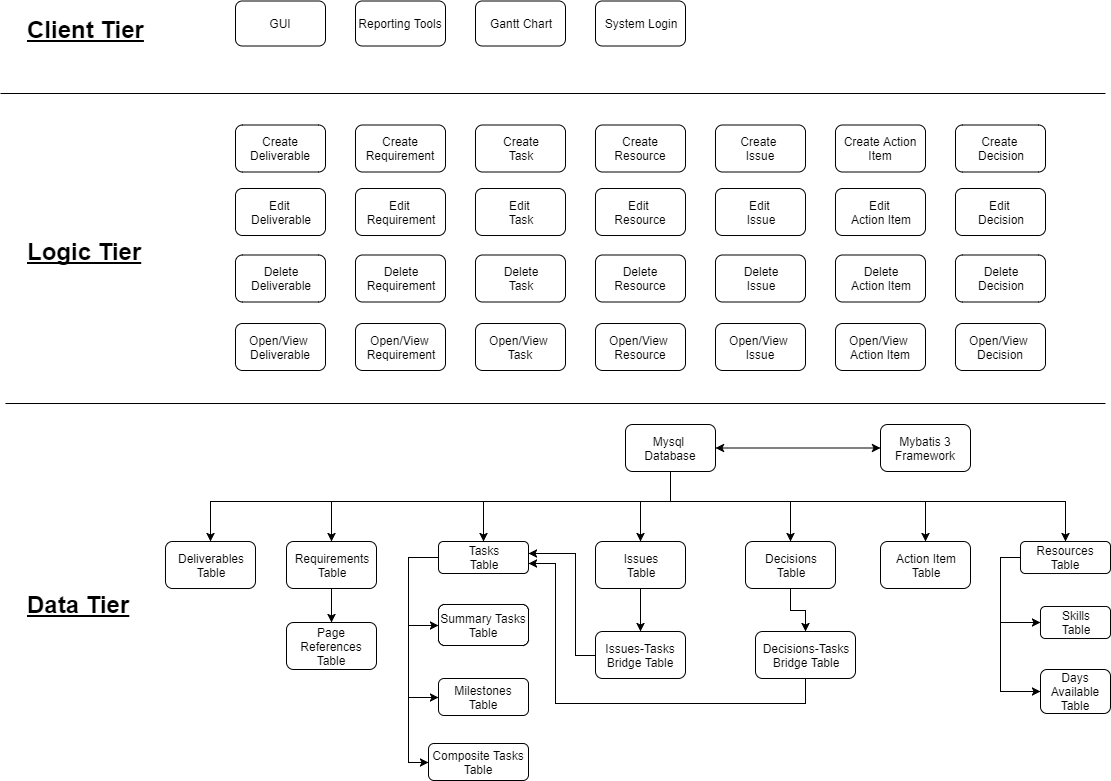
* + 1. **Decisions:**



* 1. **Use Case Diagrams:**
     1. **Deliverables:** This use case diagram shows all of the actions and tasks associated with the actions of creating a deliverable object. The figure on the left represents the user, and the dotted arrows represent the actions they can take. The figure on the right represents the program and the internal tasks it performs represented by the solid lined arrows. Note that the program mostly gives feedback to the user and automatically checks their actions for validity. The first step is for the user to begin the deliverable creation process by pressing the create option. Then they have several other options to choose from after that. Some are required for the entire process to finish, while others aren’t. The last use case to finalize the whole process and add a deliverable to the database is to save changes, then the program will automatically pass in all of the data to a java object (given all fields are valid) and then feed that data into the database. A list and description of the other tasks are as follows:
* **Input Name (Required):** A box text field component in the creation GUI where the user can type in the name of the deliverable. The validity is automatically checked by the program, then it is either set or not based on the validity.
* **Input Description (Not Required):** A box text field component in the creation GUI where the user can type in a description for the deliverable. The validity is automatically checked by the program, then it is either set or not based on the validity.
* **Input Due Date (Required):** A component of the GUI with 3 box fields, each input for the year, month and day in that specific order. In a future release, an interact able calendar will also allow the user to select a date. The date will then be automatically validated by the program and set if it is valid.
* **Allocate/Deallocate Requirements (Required):** These two actions belong to the same GUI component as they are a part of the same requirements table. The user has the option to allocate a requirement to the deliverable, which will open up a separate requirement creation menu for them. When the requirement creation process is finished that item is then allocated to the deliverable. The user can also deallocate a requirement, which will display a list of already allocated requirements to select from within the GUI component. The user can select multiple requirements, then proceed with deleting them. Note that deallocation also deletes the requirement from the database.
* **Allocate/Deallocate Tasks (Required):** This functions identically to the associations of requirements to deliverables. Tasks can be created and allocated or removed.



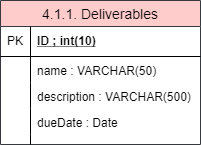
* + 1. **Requirements:**
    2. **Tasks:**
    3. **Resources**
    4. **Issues:**
    5. **Action Items:**
    6. **Decisions:**
  1. **Three Tier Diagram:**



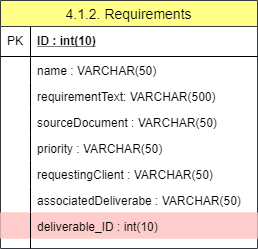
1. **GUI Prototype**
2. **Detailed Design**

**4.1** **Entity Relationship Diagrams**: The following diagrams were all created to demonstrate the structure of the backend database system. The relationships are fairly intricate and involve many entities and connections. The entire database system design would not fit onto a single page with all the attribute details of the entities. Instead, each entity has been split up into its own image with all of its detailed attributes. Entities are numbered at the top so they can be easily referenced and traced between the full system diagram and its individual images. Entities which are colored grey are either bridge-tables *(used for mapping many-to-many relationships between two entities)*, or helper tables that supply the information necessary for tables to have a list component. There are a couple custom types used as the attributes which are specific to the Mysql language. They function nearly identical to their java object counterparts.

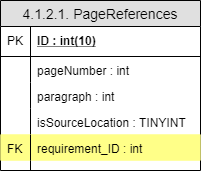
* **VARCHAR():** The string raw type for Mysql. The number inside the parenthesis indicates the length of the string.
* **Date:** The Mysql Date object which functions exactly like the Java Date object, but without the timestamps.
* **Time:** An individual timestamp using the standard GMT format.
  + 1. **Deliverables Entity Table**



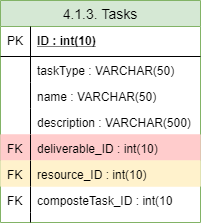
* + 1. **Requirements Entity Table**



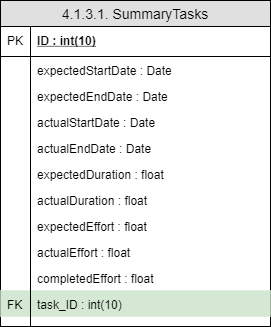
* + - 1. **Page References Entity Table**



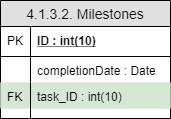
* + 1. **Tasks Entity Table**



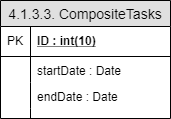
* + - 1. **Summary Tasks Entity Table**



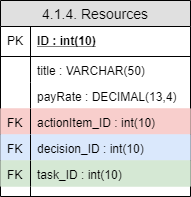
* + - 1. **Milestones Entity Table**



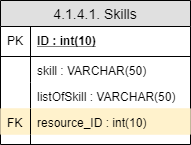
* + - 1. **Composite Tasks Entity Table**



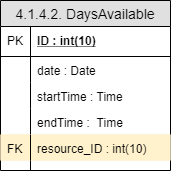
* + 1. **Resources Entity Table**



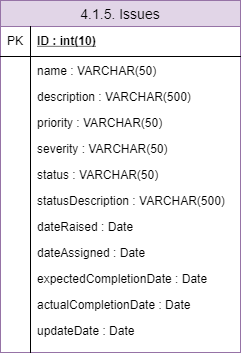
* + - 1. **Skills Entity Table**



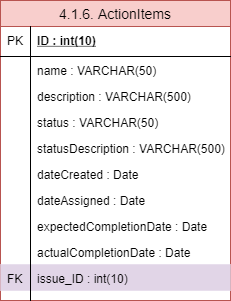
* + - 1. **Days Available Entity Table**



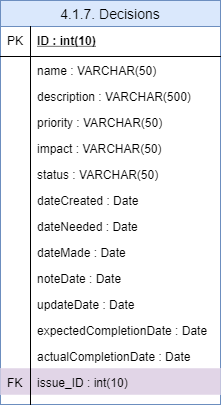
* + 1. **Issues Entity Table**



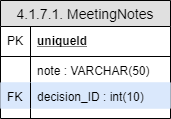
* + 1. **Action Items Entity Table**



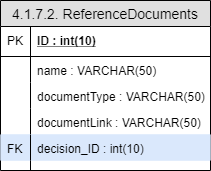
* + 1. **Decisions Entity Table**



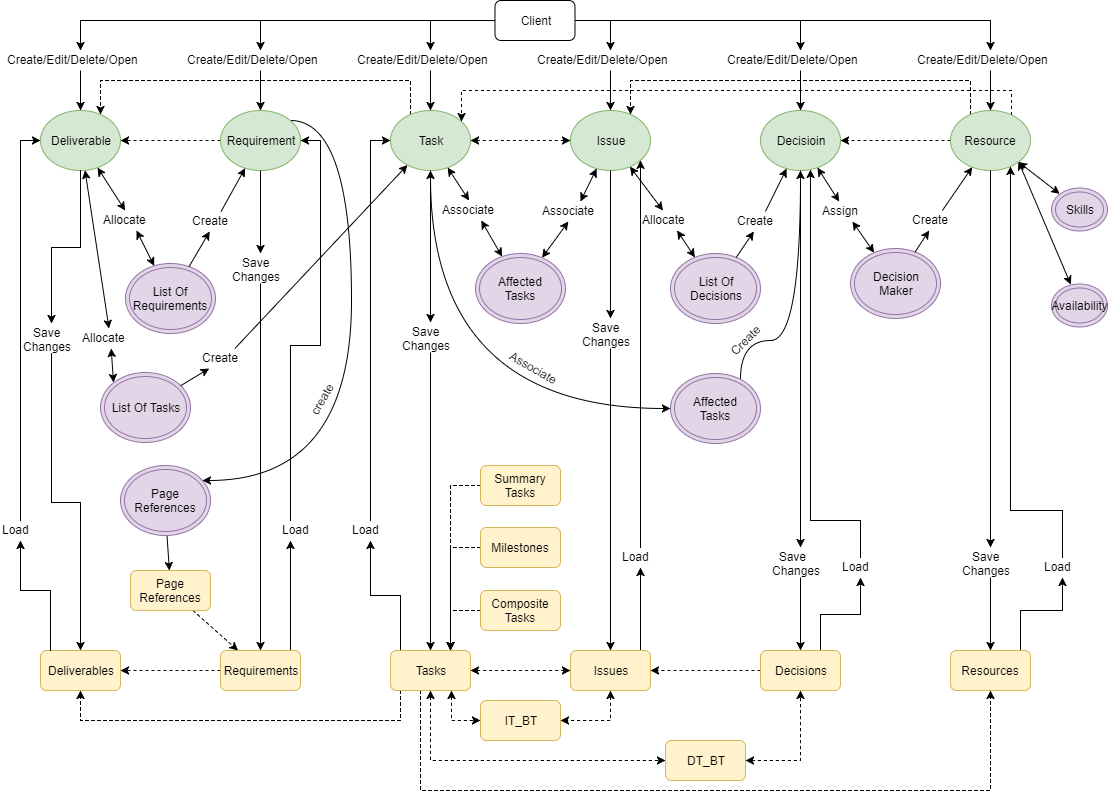
* + - 1. **MeetingNotes Entity Table**



* + - 1. **ReferenceDocuments Table**



* 1. **Data Flow Diagram:**



1. **Test Cases**

Test case tables were constructed to address each of the clients’ requirement specifications. These test cases are directly related to the implied/derived requirements. Since the size of the tables is too large to import into the report document, links will be given to each of the necessary tables. **All files can be found in the “Test Cases” directory. Every functional requirement has its own test case table document.**

1. **Requirements Cross Reference Matrix**
2. **Code Summary**