Table of Contents

[1) Project Title 2](#_Toc89708006)

[2) Introduction / Overview 2](#_Toc89708007)

[3) SRS (Software Requirements Specification) 3](#_Toc89708008)

[a. C-Requirements (Overview of customer requirements) 3](#_Toc89708009)

[b. D-Requirements (Detailed statement of User Requirements) 4](#_Toc89708010)

[4) Software Design (SD) 5](#_Toc89708011)

[a. Architectural Design / System Design 5](#_Toc89708012)

[b. System Design / component-level design 5](#_Toc89708013)

[5) Verification and Validation (V&V) 6](#_Toc89708014)

[6) Software Quality Assurance (SQA) 6](#_Toc89708015)

[7) Project Metrics 6](#_Toc89708016)

[8) Design Patterns Used or Proposed 6](#_Toc89708017)

[9) Possible or Actual Refactoring 6](#_Toc89708018)

[10) Appendices: 6](#_Toc89708019)

# Project Title

Gotta Collect ‘em All!

A simple system for tracking personal collections of items.

# Introduction / Overview

The “Gotta Collect ‘em All” system serves as a repository for items in a personal collection. A user can add any type of item into the system, view added items, and edit/delete items. The system will be accessible via a Graphical User Interface (GUI).

One example would be tracking a baseball card collection. They can add a card with at least a title (name of card) and quantity. Other important information describing the card can also be added like the print year and condition. Items can be viewed, added, removed, and updated.

# SRS (Software Requirements Specification)

## C-Requirements (Overview of customer requirements)

* + 1. Introduction

The Gotta Collect ‘Em All system will help users mange their collections.

* + 1. Conceptual model (High-level system diagram)

<https://app.creately.com/diagram/NLgCLmbznkA/edit>

Diagram

Description automatically generated

* + 1. Overall description

The system will allow users to manage their collections. The unique thing about this application is that it can handle multiple types of collections. The system doesn’t need any prior knowledge of the attributes of a collections. The User will be responsible for all details of a collection.

Each collection will have a unique identifier as to separate it from other collections. Inside each collection, each item can have similar attributes as the other items or unique information.

The system will allow an user to perform basic functionality on their collection. This includes, viewing items, inserting new items, updating existing items and deleting existing items.

## D-Requirements (Detailed statement of User Requirements)

* + 1. Functional requirements
       1. The system has the capability of tracking multiple types of collections. i.e. Baseball cards, pottery, or spoon collections.
       2. Each collection will be uniquely identified.
       3. Each item inside a collection can have different types of information.
       4. At a minimum each item in a collection will have a name identifier and a quantity describing the quantity of an individual item.
       5. The system will track the owner of a collection.
       6. The system will have a graphical user interface allowing a user to interact with the system.
       7. Users can view their entire collection at once.
       8. Users can insert new items.
       9. Users can delete existing items \*Future Requirement
       10. Users can update existing items \*Future Requirement
       11. The current date/time will be saved to track the last time an individual item in a collection was modified.
    2. Non-functional requirements
       1. Performance, Capacity, Reliability
          1. Standard performance requirements will be followed for user response time. Each action by the user will not take more than 1 second before an action is completed.
          2. It is acknowledged that the system will be limited on how many items it can store depending on the size of the physical space where the database is installed.
       2. Error handling, backups, invalid data handling
          1. \*NOTE\* Specific error codes and descriptions for specific conditions to be determined at a later date.
          2. Each system error will be displayed back to the user with a descriptive message that informs the user on how to fix or bypass the error.
       3. Interface (API & GUI)
          1. The system will have a Graphical User Interface to allow all user functionality.
       4. Constraints, limitations
    3. Inverse Requirements
       1. The system will not contain images of items.
       2. The system will not check for accuracy of the items entered.
       3. No duplicate validation will be performed.
       4. Users will not be required to authenticate to use the system.

# Software Design (SD)

## Architectural Design / System Design

Create

User Interface

Delete

Update

Read

Collection Orchestrator

Collection Database

## System Design / component-level design

* + 1. GUI (Graphical User Interface):

The graphical user interface will act as a façade allowing the user to input/view the data.

The GUI will make API calls to the back end “Collection Orchestrator” application for processing of the data.

Required fields for the query of a user’s collection:

* Input:
  + User/Owner
* Output:
  + User/Owner
  + Item Name
  + Quantity
  + Any other field related to an item

Fields to be ignored when viewing an entire collection:

* Last Updated date/time
* Item Unique hash id that is auto generated by the database

Required fields to be listed on the screen for an Insert of an item to a user’s collection:

* Item Name
* Quantity
  + 1. Collection Database:

Since the system requires different types of data with dynamic labels/columns, a MongoDB solution is preferred as the database technology. More information can be found on their official website: <https://www.mongodb.com>

Collections can share the same ‘document’. However, with an increase of data and users, it might be necessary to create a ‘document’ per individual user.

The database doesn’t require any initial scheme, however the Collection Orchestrator will enforce a few required fields

1. Owner
2. Item Name
3. Quantity
4. Update Timestamp (in UTC time)

* + 1. Collection Orchestrator API diagram:

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# Verification and Validation (V&V)

## Software Integrity level

**1** - Software must execute correctly or intended function will not be realized, causing negligible consequences.

## Purpose

## Referenced Documents

## Definitions

## V&V Overview (Organization, Schedule, . . .)

## V&V Processes

|  |  |  |
| --- | --- | --- |
| **Tasks** | **Input** | **Output** |
| Evaluate *proposed software changes* | SVVP proposed changes | Updated SVVP |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **V&V Activity** | **Requirements** | **Design** |  |  |
| Acceptance *Test Support* | N/A | N/A |  |  |
| Component V&V Test Plan | N/A | N/A |  |  |
| Integration V&V Test Plan | N/A | Yes |  |  |
| Risk Analysis | N/A | N/A |  |  |
| Security Analysis | N/A | N/A |  |  |
| S/W Design Evaluation | N/A | Yes |  |  |
| S/W Requirements Evaluate | Yes | N/A |  |  |

## V&V Reporting Requirements

## V&V Administrative Requirements

## V&V Test Documentation Requirements

# Software Quality Assurance (SQA)

# Project Metrics

## FUNCTION POINTS

Function Points, FP = CT X [ 0.65 + 0.01 X ∑Fi ]

Fi = SUM of the following from 0-5 where 0 is No Influence and 5 is Essential

1. Does the system require reliable backup and recovery? 1

2. Are data communications required? 3

3. Are there distributed processing functions? 0

4. Is performance critical? 0

5. Will the system run in an existing heavily utilized operational environment? 0

6. Does the system require on-line data entry? 5

7. Does the on-line data entry require input transaction to be built over multiple screens or operations? 0

8. Are the master files updated on-line? 5

9. Are the inputs, outputs, files, or inquiries complex? 1

10. Is the internal processing complex? 1

11. Is the code designed to be reusable? 3

12. Are conversion and installation included in the design? 1

13. Is the system designed for multiple installations in different organizations? 0

14. Is the application designed to facilitate change and ease of use by the user? 1

Fi = 21

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Weighting Factor | | |  |
| Parameter | Count |  | Simple | Average | Complex |  |
| Number of User Inputs | 3 | X | **3** | 4 | 6 | 9 |
| Number of User Outputs | 1 | X | **4** | 5 | 7 | 4 |
| Number of User Inquiries | 1 | X | **3** | 4 | 6 | 3 |
| Number of Files | 1 | X | **7** | 10 | 15 | 7 |
| Number of External Interfaces | 0 | X | **5** | 7 | 10 | 0 |
|  |  |  | Count Total, CT = | | | 23 |

**Total FP = 19.78**

**Calculate Lines of code based on FP = 593.4 (Actual ~ 320)**

**Calculated People Months = 1.38 (Actual ~.25)**

**Calculate Dev time in months = 2.8 (Actual ~.25)**

# Design Patterns Used or Proposed

Microservice Pattern – Software as a Service - <https://12factor.net>

# Possible or Actual Refactoring

# Appendices: