Contents

1 Analysis

1.1 Problem Identification

1.1.1 Problem Description

Popular inventory management solutions are relatively expensive, and may be out of reach for individuals or small schools. Inventory systems have numerous benefits for businesses and individuals alike; a business may choose to track their supply levels where an individual may wish to catalogue their DVD collection.

My goal is to create a web-based application aimed at both businesses and individuals to manage inventory, with additional modern features such as automatic item re-ordering when stocks are running low.

Traditional inventory management solutions are typically single-user at best, whereas I intend to create a multi-user, collaborative environment.

In my view, an inventory system should be:

- Easy for end users to use.
- Cross platform
- Performant interface
- Efficient in terms of adding data
- Allow for easy cataloguing of inventory
- Allow for item scanning using QR codes / barcodes
- Be able to source data from external sources
- Support both consumable and non-consumable goods.

1.1.2 Stakeholders

Stakeholder requirements are further discussed for each stakeholder in the Stakeholder Requirements section.

Stakeholder	Description	Requirements	Capability
Claire Foley	Senior Leadership Team at The Village Prep School	Ability to manage library books. Admin and supervision of other users carrying out librarian tasks	Well-versed in computer use, at least when it comes to intuitive and well designed interfaces. Would struggle with a non-intuitive interface design.
Ella	”Head Librarian” (Pupil) at the Village Prep School	Ability to check in and out library books. User of the system; requires interface that is appropriate for her age.	Beginner user of technology, proficient in mobile applications on phones and tablets only. Rarely uses a laptop or desktop computer.
Generic Gear Rental Shop	Photography gear for hire business	Ability to manage business inventory in a fast and efficient manner.	Proficient with computers.

1.1.3 Why is it suitable to a computational solution?

1.2 Investigation

1.2.1 Preparation for interview

Question Set

- What would you consider your skill level to be regarding technology?
- Do you currently have a way to manage inventory?
- If so, what is your current solution?
- What aspects of this solution do you like?
- What aspects of this solution do you dislike?
- What features would you **require** in a custom solution?
- What features would **enhance** your experience?

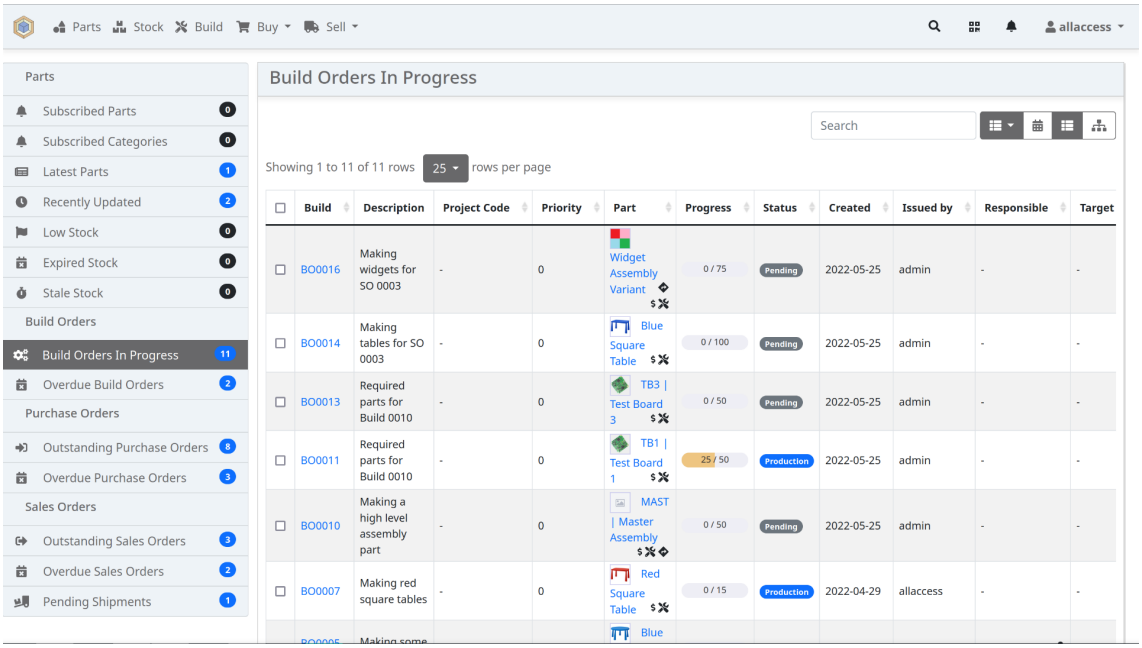
1.2.2 Interviews

1.2.3 Summary of interviews

1.3 Research

1.3.1 Existing similar solutions

InvenTree `https://inventree.org/`



Overview
InvenTree is an **open-source** inventory management system, providing *low level stock control and part tracking*. It uses a Python/Django database backend and provides both a **web-based interface** as well as a REST API for interacting with other services. InvenTree also has a powerful plugin system for custom applications and other extensions.

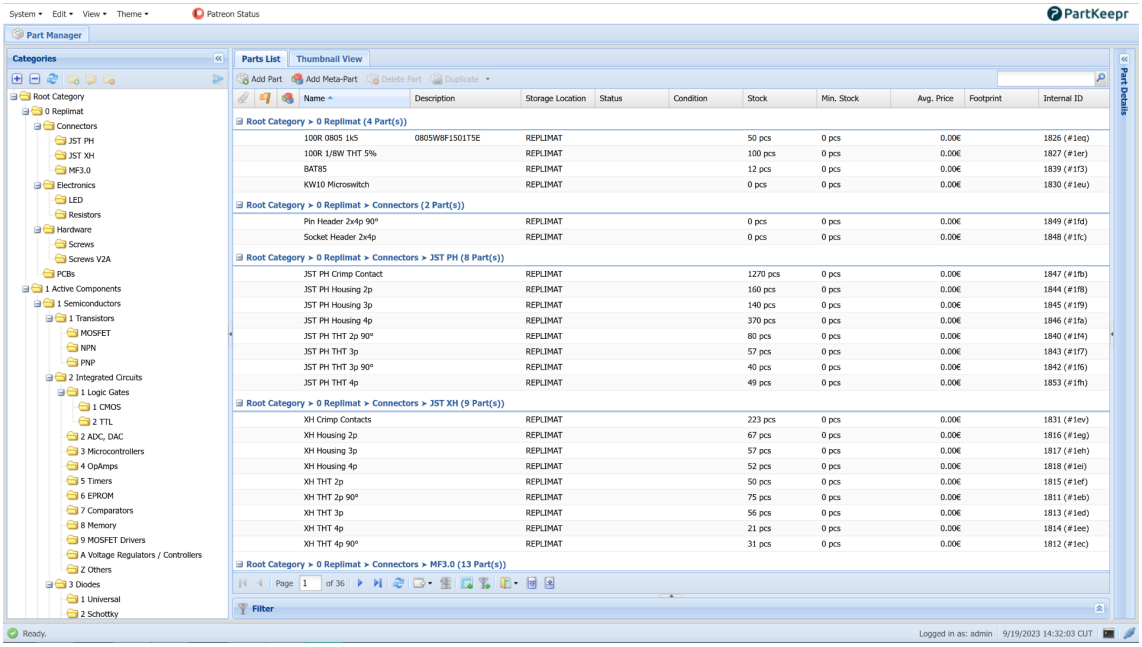
Parts applicable to my solution

- Web-based application
  The application will be web-based.
- Modern, Relatively simple user interface
  InvenTree offers a relatively simple and intuitive user interface.

Parts <u>not</u> applicable to my solution

- Stock control and part tracking specific features
  I am looking to implement a system that is capable of being far more generalized than just part tracking, although the system will have features for library book tracking.

PartKeepr `https://partkeepr.org/`



Overview

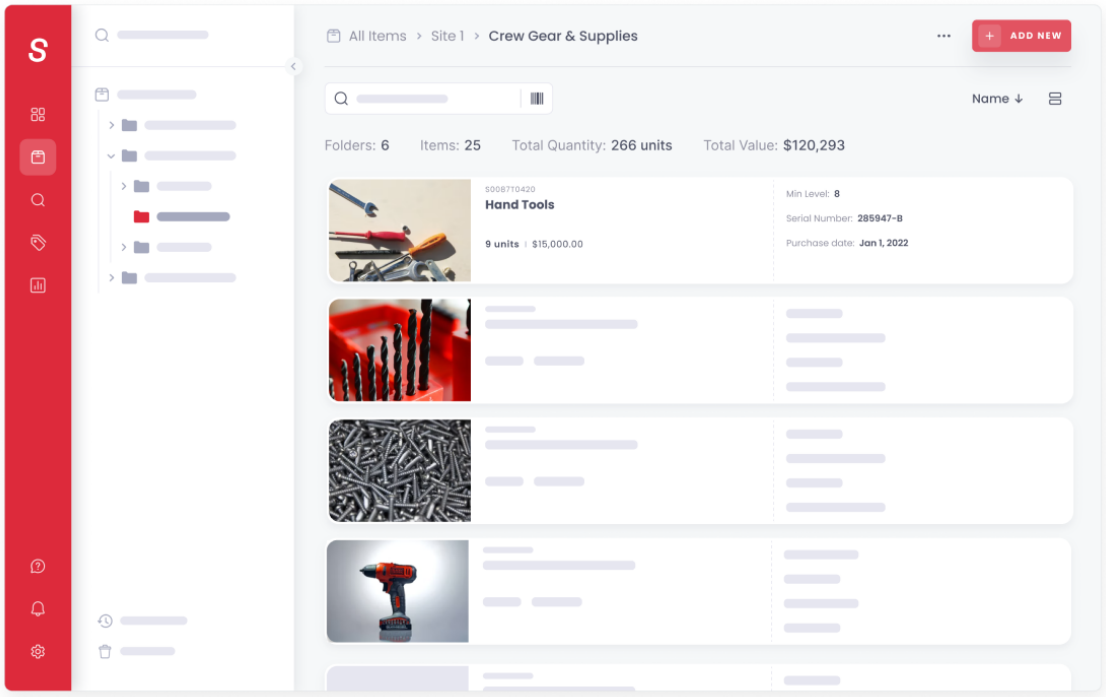
PartKeepr is an open-source inventory management system with a focus on electronic components. It is designed around four main principles:

- Fast Part Searching
- Ability to add complete part database
- Keeping track of stock
- Ease of use

Parts applicable to my solution

Like PartKeepr, I hope to implement a web-based interface. However, I am using a different approach as my solution will not be tailored specifically to electronic components.

Sortly `https://www.sortly.com/solutions/inventory-management-software/`



Overview
Sortly is a proprietary cloud-based inventory management system with a focus on small businesses and inviduals.

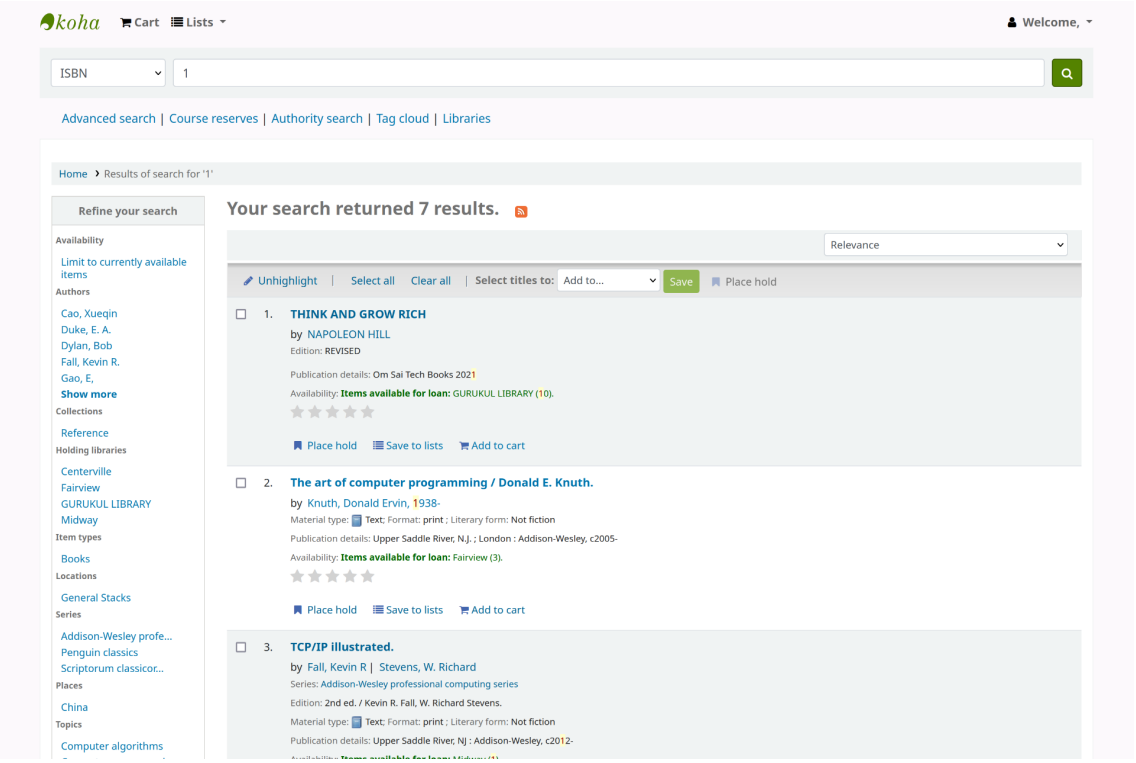
It has two plans available, an always free plan with limited functionality and a paid plan will a more complete feature-set.

Parts applicable to my solution
I hope to implement the following features from Sortly:

- Web based interface
  - Allows for easy access.
- Barcode support
  - Allows end users to print off QR codes to stick to items
  - Which can be scanned in-app to easily perform actions on the item.
- Real-time reporting insights
  - Allows for added insight into usage patterns for particular units.

Koha

`https://koha-community.org/`



Overview

Parts applicable to my solution

1.3.2 Features to be incorporated into solution

1.3.3 Limitations of the solution

1.3.4 Feedback from stakeholders

1.4 Requirements

1.4.1 Stakeholder requirements

1.4.2 Software and hardware requirements

System Requirements

Hardware	Justification
Laptop/Desktop Keyboard and Pointing Device (eg. Mouse)	For desktop or laptop computer users, a suitable input device is required in order to interact with the software. A pointing device (a mouse) is necessary in order to interact with the user interface, to perform actions such as clicking buttons, icons, and opening menus. A keyboard will be used to manually input data into the system.
Tablet Device Touchscreen	For tablet users, it would be impractical to expect the user to have access to a keyboard and or pointing device. Therefore, we must design the system to accept inputs from a touchscreen. This will be easier to use and more intuitive for tablet users. The touchscreen will be used to input data into the system and to interact with the user interface.
Dual-Core Processor (x86, ARM, RISCV architectures)	A modern processor with sufficient resources to run an up-to-date web browser such as Chrome, Edge or Firefox is required in order to access the web-based interface.
2GB of RAM	Sufficient RAM is required to run the web browser, which can be a memory intensive task.
Monitor	To display the user interface.
Network Interface Card (NIC)	A Network Interface Card, or NIC, is required for the computer to be connected to a network, such as the Internet. This is required as the web interface will be hosted on a domain and server that is external to the user, that is to say, not on their local network.
***Optional**: Wireless Network Adapter*	*A Wireless Network Adapter is an optional requirement, it will allow the user to connect to a wireless network in order to access the network or Internet so that they can access the external user interface.*
***Optional**: Camera*	*A Camera is an optional requirement; devices with cameras will be able to scan barcodes or QR codes corresponding to inventory items and easily perform actions on them.*

Software Requirements

Talk about why I don't need much software since dependencies hosted on the server.

Software	Justification
Operating System *(Windows, MacOS, Linux, ChromeOS, iOS, iPadOS, Android)*	An operating system is required in order to run the web browser necessary to access the interface.
A web browser *(eg. Chrome, Firefox, Microsoft Edge, Safari)*	A web browser is necessary to access the interface as it will be primarily a web application.

1.4.3 Success requirements

The overall objectives for the system.
To measure the overall effectiveness of the system, targets must be set before writing the program. These targets will help in the evaluation stage to determine weather our objectives have been met. These objectives will be **SMART**, i.e:

- **Specific**
  What objective needs to be accomplished?
- **Measurable**
  How can we quantify this objective?
  How will the success of this objective be measured? (quantitatively or qualitatively)
- **Achievable**
  Is this objective achievable and realistic? If so, how to you plan to achieve them?
- **Relevant**
  How does this objective benefit the end-users of this application as a whole?
  Why has this goal been set?
- **Timely**
  Can this objective be completed within an appropriate time frame?
  At what stage in the software development lifecycle will you start implementing this goal?
  In which order will any sub-objectives be completed?

The Project's SMART Objectives

1. **To produce a solution for cataloguing a school library and recording users and books borrowed**
   At the end of the project, I will evaluate against my success criteria and determine weather this objective has been met. On the software side, I will be using React, Expo and PostgreSQL. This objective will be the main objective for this project. This objective must be completed by March 2024.
2. **To produce a solution including a database that can store details of books, borrowers, loans and returns**
3. **To produce an intuitive and easy to use solution**
   I will evaluate my success on this objective by having a new user without any prior training or advice use the system and try to carry out a number of tasks without any assistance. If the user is able to successfully complete the tasks I will consider the system to be intuitive and easy to use and therefore this objective satisfied. To achieve this I will design my system to have a consistent layout based on **Material Design 2**, (`https://m2.material.io/`) the design language used by Google products and many apps running on the Android operating system. I will also use language that is a) appropriate for the situation the product will be deployed in (with young children) and b) easy to understand (so that children can interact with the system) I will also use meaningful error messages so that the user has a clear understanding of the problem that has occurred. This objective will benefit the end user as a intuitive and easy to use solution is critical to the usefullness of the project. If the end product is not easy to use, it is less likey to be used and accepted by my stakeholders. This objective will be worked on during the development process, and so will be completed by the time development concludes. I will mock-up a version of the user interface in the design stage and will continuously iterate on the user interface during development.
4. **To produce a solution that features a fully searchable catalogue**
5. **To produce a solution that features reporting for overdue and/or lost books**
6. **To produce a solution that includes a curated "suggested reading list" for each borrower**
7. **To produce a solution containing a user interface that can be accessed via a mobile device**

2 Design

2.1 User Interface Design

2.1.1 Usability Features

2.1.2 Feedback from stakeholder

2.2 Modular breakdown

2.3 Algorithms

2.4 Data Dictionary

2.5 Inputs and outputs

2.6 Validation

2.7 Testing

2.7.1 Methods

2.7.2 Test Plan

3 Implementation

3.1 First Iteration — Initial Backend and Database

3.1.1 Introduction

In this sprint I will work on the backend service. This service will provide an interface for the frontend to talk to the database via an API (Application Programming Interface). I am writing the backend in **Go**. Go is a performant, statically typed high level language designed by Google. Is is frequently used for backend development thanks to it's performance and memory safety. I am going to use GraphQL as the query language for the frontend to interact with the backend.

GraphQL is an open-source query and manipulation language designed for use in APIs. (Application Programming Interface). The backend will serve as an API which will interface with my database. I choose to use GraphQL as it is better suited for larger, more complex data sources, and supports querying for multiple different types of data at once, unlike REST. It is also something I was interested in learning more about as I have not designed a system using it before.

TODO: explain what a graphql mutation is (it's a function)

3.1.2 User account creation

The first feature I decided to work on was user account creation. This would involve asking the user for an email address, name and password, before validating it and inserting it into the database. In addition, at a later stage, validation must be performed in order to ensure that:

- The user email is not already in use
- The generated user ID is unique and not already in use

For this early stage of development, I decided to use an SQLite database to make things easier. I can easily switch this to PostgreSQL as specified in my design doc later.

When a GraphQL mutation is executed to create a new user, a function "CreateUser" is called, which is passed any input from the query and a connection to the database.

My first version of this function was as follows:

```go
// CreateUser is the resolver for the createUser field.
func (r *mutationResolver) CreateUser(ctx context.Context, input model.NewUser) (*model.User, error) {
    // Create the user struct
    user := structs.User{FirstName: input.FirstName, LastName: input.LastName, Email: input.Email}

    // Generate a user ID
    user.ID = uuid.New()

    // Create the database entry
    r.db.Create(&user)

    return &model.User{
        ID: user.ID.String(),
        FirstName: user.FirstName,
        LastName: user.LastName,
        Email: user.Email
    }, nil
```

(NEXT UP: UUID GEN)

Unique ID generation

I decided to use a **for loop** to continuously generate UUIDs (Universal Unique Identifier) to be used as a potential User ID. I then perform validation on the UUID to ensure that it is not already in use. This can be done with the following code:

```
func (r *mutationResolver) CreateUser(ctx context.Context, input model.NewUser) (*structs.User, error) {

[..]

    isFreeUuid := false
    for !isFreeUuid {
        // Generate a UUID for the user id.
        user.ID = uuid.New()
        // Check that the UUID has not been used already
        // If true, it will break out of this for loop and continue.
        isFreeUuid = util.IsUuidFree[structs.User](r.db, user.ID, &structs.User{})
    }
```

In order to achieve this and reduce code duplication across different functions, I created a ”IsUuidFree” utility function. Here is the initial version of this function.

```
func IsUuidFree[T any](db *gorm.DB, id uuid.UUID, obj *T) bool {
       err := db.Model(obj).Select("id").Where("id == ?", id.String()).First(&obj).Error
       if errors.Is(err, gorm.ErrRecordNotFound) {
              // Record not found, so user id is free
              return true
           } else {
              // Record was returned successfully, therefore the user exists
              return false
           }
    }
```

This function makes use of **generics**. As per the Go docs:

> *With generics, you can declare and use functions or types that are written to work with any of a set of types provided by calling code.*

To simplify, generics mean that I can pass any struct (**T**) to the function and the function will use that type for the obj parameter. For example, if I call the function with:

```
util.IsUuidFree[structs.User](r.db, user.ID, &structs.User{})
```

Then T is set to the type `structs.User`, allowing me to pass an object of type `structs.User` as the third parameter.

GORM works by defining a struct to query for (corresponding to a table in the database, in this case the *”users”* table). We can then perform SQL actions on it, such as Select.

Therefore, this gorm DB call is the equivalent of:

```
SELECT id FROM users WHERE id == ? VALUES ("uuid-goes-here")
```

TODO: this was modified, show where it was modified later on?

para

4 Testing

5 Evaluation