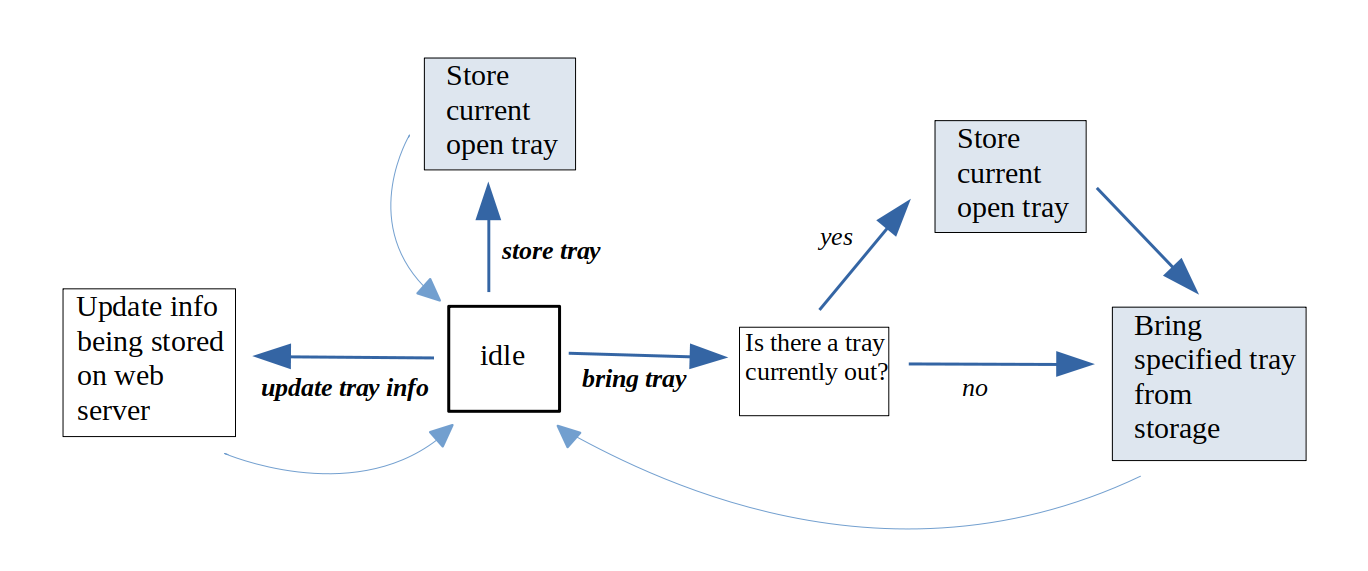
# Requirements Document

Our system THEOStore will provide convenient autonomous storage for people who may suffer from mobility issues or other ailments. This document will provide all the requirements and features of the software implementation which will help produce the final product. We opted for a refined requirement document due to the high-level description and other parts already been written in the project plan, instead of repeating ourselves we went into more depth about requirements.

## Functional Requirements

|  |  |  |
| --- | --- | --- |
| **Functional Requirement** | **Description** | **More Specific Requirements** |
| **User Requests** | Allow users to request to store/retrieve items either through the app or voice command | When using voice commands, the app will convert the correct instruction requested then send the details to THEOStore of what needs to be accomplished.  Once item is retrieved the app shall allow the user to instruct THEOStore to put the tray back |
| **Connect to Webserver** | The app must connect to the Webserver so it can communicate with the hardware aspects of the system (THEOStore robot) | Once the app is given instructions from the user it needs to send commands to THEOStore describing what needs to be retrieved, stored, etc. The app will also use the database for this step as it contains information about where items are stored, etc. |
| **Database** | App should link to a database that contains information about each item that is stored. For example, which tray its stored and where the tray is located | App will allow users to manually label trays (name and/or summary) and app will link user labels to the item database |
| **Tray photos** | App will store photos THEOStore takes of tray contents and allow users to view the images | The image will be processed to estimate how full trays are so that when storing new items THEOStore won’t try to put an item in a full tray. When browsing the list of trays the app will display how full each one is. |
| **UI Design** | The user interface will be designed with accessibility and ease in mind. | The user interface must have sufficient contrast between foreground and background  Users will register an account which they can access when using the system  Clear and consistent navigation options  Colours will be suitable with colour blind and visually impaired users  A simple layout with help functions |
| **Stretch Goals** | The system will be able to predict tray contents from an image using object recognition. More intelligent voice control functionality may also be something to aim for. | As basic voice recognition will be implemented on one of the main requirements a stretch goal was to improve this to recognise more commands than just basic ones. |

**Further details on some Functional Requirements:**



**Database:**

The database is where all information needed would be permanently stored. Each entry in a database would include information in the following columns:

|  |  |  |
| --- | --- | --- |
| **Column name** | **Description** | **Data type** |
| tray name | Name / ID of the tray | text |
| tray info | Summary of the trays contents (could be empty) | text |
| currently out | Is the tray at the accessing point or on the shelf? | Boolean |
| capacity | How much space is available in the tray? Calculated from the image of the tray contents. | Probably some restricted choice is best, but could be a numerical value |
| path to image | Images are stored on the web server; we keep track of which image is for each tray | text |
| position data | Data on a tray’s physical location on the shelf | ??? - *maybe multiple columns are needed* |

The robot would interact with the database to get the information on tray names and their positions on the shelf. The phone app would also request data from the database - specifically information like tray info, currently out and capacity. The phone app would also need to request images stored on the web server.

**User Requests via voice control:**

Also note that while we would use some external web service to convert speech audio to text, some basic handling of the speech text would be done in the phone app to convert it to the appropriate command meant by the user. It wouldn’t be hard to convert input text like “bring me tray X” into the appropriate bring tray command to get tray X – but the extent to which the system handles varying sentence structure as input is a stretch goal of the project.

**Robot Controller Functionality:**

*Bring specified tray from storage:*

- We know that there is no tray currently out (so we have space to move our tray to)

- Move the specified tray from its position on the shelf down to the accessing point.

- We keep track of the fact that there is now a tray out of the shelf

*Store current open tray:*

- We know there is a tray out (note that ***store tray*** would only be sent from the phone app if a tray is out)

- Have the camera take a photo of the tray contents, and update this trays image which is stored on the web server

- Then move the tray back to its position on the shelf

- We keep track of the fact that there is now no tray out of the shelf

The robot controller would control how specific parts of the robot move, and this would require knowledge on physical dimensions of the robot and positions of trays. Information on the robot state and a mapping from tray name to tray position could be modelled using classes like the following:

class Tray {

name: e.g. ‘A1’, trays are maybe in some sort of coordinate system

position: data on its physical position on the shelf, each tray has its own position where it stays

}

class Robot {

open\_tray: instance of Tray, or Null

position\_data: info on physical state of the robot *– can’t do any meaningful planning here*   *until a robot design is completed*

}

***Estimating tray capacity from tray image:***

A brief overview of how it may need to work...

Given that trays are brought to the same position each time, one could take a reference photo of an empty tray, and then compare other photos of trays to this reference photo.

A simple approach could be to first convert images to grayscale images (colours don’t matter), and then subtract the reference photo from a tray image. The resulting ‘difference of images’ would only have shapes where items are stored in the tray.

Then one could find the bounding shape (or shapes) of all items being stored, and then find the total area of these (this should be fairly straight forward using a library like OpenCV). Trays which are fuller will have a larger area – all that is then needed is a mapping of this area to some ‘capacity’ value.

**Phone App Functionality:**

**Find Tray**

The searching for a tray functionality would be handled in the phone app using the text stored as tray summaries. Consider the following description of how searching for a tray could work:

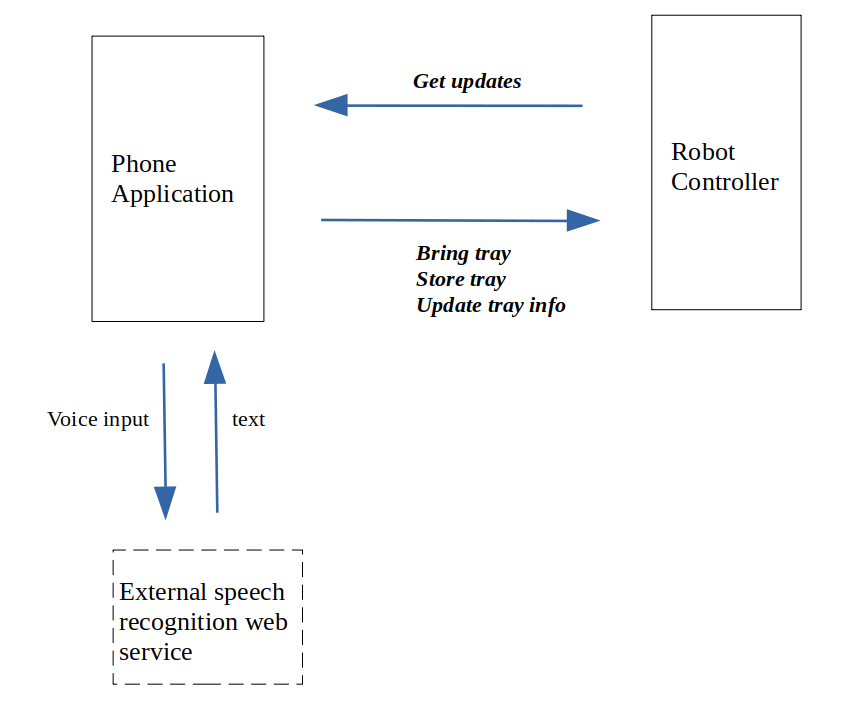
*Find tray:*

* Get input text to search for (entered into a search box or via voice control)
* Get a list of meaningful words from the input text (remove standard English words ‘a’, ‘the’ etc.)
* For each tray, count the number of matching words there are in the tray summary and search input.
* If there are no matching words at all, none of the tray summaries describe what was searched for – search failed.
* Otherwise, the tray which had the most matching words is the most appropriate tray to bring – so the app displays this tray to the user.

**Bringing a random tray**

When the user wishes to bring a tray, this ‘bring tray’ command is always given to the robot controller with a specified tray. However, in the phone app on the main screen the user could request a random tray. Then the app would find any empty tray (or pick the emptiest tray) by considering the tray capacity data stored on the systems database (described more later), and then it would issue the ‘bring tray’instruction to the robot.

## External Interface Requirements



* User interfaces
  + User interface design details. Add a help button on each screen, shortcut keys, screen layout, accommodation for visually impaired etc.
* Software interfaces
  + The database contains information about each item is an external software interface. This is where all the data is kept concerning which tray an item is stored in and where the tray is located in the storage area. Without the database THEOStore will have no way of remembering where everything is stored.
* Hardware interfaces
  + The interface connection between software and hardware is the Webserver. The Webserver needs to be working as this is the communication between the app and THEOStore, without this connection users won’t be able to request tasks from THEO.

## System Features

* Usability first
* Push notifications
* Feedback system
* Social Integration
* Augmented reality (camera attached to the robot to give live streaming of their object being stacked)
* Advanced Analytics (which item they use the most, is it good for them - source of inspiration -> Apple Screen time feature)
* One click contacting - Customer care
* QR/Barcode Scanner integration
* Security - privacy of items maintained, placement of malware, stealing customer data/fraud
* Search Engine - To search the list of items
* Responsive Mobile App Design for Varying Screen Sizes
* Ability to work offline
* Speed
* Good image resolution
* Good (bold) colour schemes of the app
* Updates - to improve the app

## Non-functional Requirements



|  |  |  |
| --- | --- | --- |
| **Non-Functional Requirements** | **Description** | **More specific requirements** |
| **Performance and scalability** | How fast does the system return results? How much will this performance change with higher workloads? | Backup of items/objects will ensue with proper security of the date |
| **Portability and compatibility** | Which hardware, operating systems, browsers, and their versions does the software run on? Does it conflict with other applications and processes within these environments? | 1. Software installed must be compatible with its firewall 2. Cross platform operations allowed |
| **Reliability, Availability, and maintainability** | How often does the system experience critical failures? and how much time is it available to users against downtimes? | 1. 50 % maintainability rate? 2. 90% availability rate? |
| **Security** | How is the system and its data protected against attack? | Under such circumstances unauthorised access takes place |
| **Localisation** | Does the system meet local user’s requirements? | 1. Languages to access control and control the app would be English and? 2. UK text language will be used |
| **Convenience for the user to control the app** | How easy is it for the customer to use the app? | 1. Learnability - How fast is it for users to complete the main actions once they see the interface (ideally -how much time should it take in minutes - will ensue easier to use design etc) 2. Efficiency. How quickly users can reach their goals for e.g., time taken to accomplish selection of item and sending the message to the robot 3. Memorability. Can users return to the interface after some time and start efficiently working with it right away? 4. Errors. How often do users make mistakes for e.g., error rate for sending a request of stacking one item should not exceed 10 percent 5. Satisfaction. Is the design pleasant to use? From a scale of 1-10; an average of 7 should be accomplished |