YEAR 12 software design and development major work

**Part a**

**Shrey Somaiya**

Avondale school **|**

PROBLEM DEFINITION AND DISCUSSION OF RELEVANT ISSUES

# The problem

Wishlists are collections of products an individual would like purchased for them for an event such as:

* Christmas
* Birthdays
* Valentine’s Day
* Anniversaries
* Easter
* And many more occasions

Currently, there exists no easily accessible, intuitive and aesthetically pleasing software product that allows individuals to create and share their own personalised wishlists to their friends and family.

# Target audience

The target audience consists of anyone over the age of 14, as that is the age that they would most likely have their own (or school issued) laptop, and by that age are computer literate. 14 is al­so the age that teenagers begin to want more control and independence in their life, for example: having input in their presents.

# Needs of thE client:

## functionality requirements

The program must meet the following functionality requirements:

* Must be able to sort objects in a wishlist alphabetically
* Must be able to sort wishlists alphabetically
* Must be able to create a wishlist
* Must be able to save a wishlist to local computer
* Must be able to save wishlist to server
  + Using gmail as a server
* Must be able to reload and automatically add new wishlists that you own or are shared with you (found on local or on server) to screen
* Must be able to share wishlists
* Must be able to sign up
* Must be able to log in

## Compatibility & Performance requirements

The program must be able to meet the following compatibility requirements

* The program will only be able to run on Macintosh operating systems, on desktop only.
* No specific graphics card is needed.
* A keyboard or a typing interface is needed.
* The design is optimised for 1440x900 but is aimed to be dynamic and support desktop resolutions.

The program must be able to meet the following performance requirements:

* The program should run at the same speed under all network conditions/speeds. The sorting and/or searching is unreliant on the internet, and is done in the backend of the program.
* The list can only have a maximum of 100 items, so sort time(of an insertion sort) should take at absolute worst case O(n2) complexity
* The program will only be reliant on the internet when sharing, saving , or loading from the server. Otherwise the viewing and editing Wishlists will work as expected.

## Boundaries of project

The program must adhere to the following boundaries in order to maximise storage and program speed/efficiency

* Passwords must be less than 30 characters in length
* Full name must be less than 60 items.
* Lists can be no longer than 100 items.
* There can only be one user logged in at a time.
* Each user can only have a maximum of 10 lists
* Each user can only have a maximum of 100 lists shared with them
* Each list item must be less than 100 characters in length
* Each name of a list must be less than 30 characters
* Each email MUST be a gmail address.

# Discussion of existing solutions:

Ther are many existing wishlist making programs, however there are also many problems with these programs, as outlined below:

**Amazon** **Wishlists**: Web-based, addon, GUI design isn’t nice, user experience is clunky, and not a standalone application, internet reliant.

**Wishpot:** Web-based, browser plugin, clunky UI and UX, internet reliant.

**Google Docs:** Word processor-not designed to be a wishlist program, Web-based, no efficient sharing method

**Evernote:** Word processor/note taker- not designed to be a wishlist program, Web-based, no efficient sharing method, UI is centred around taking notes, premium is costly

**Wishlistr:** Web-based, only able to make one list, not easily able to view shared Wishlists.

My solution is preferred as it provides a clear and accessible UI to access, edit, view and share Wishlists. Unlike the existing solutions, my solution will not be a clunky web browser add-on. My solution is a desktop app designed for the sole purpose of creating, editing viewing and sharing wishlsits. The solution will be easily accessible at all times, and does not require internet access to view or edit Wishlists. It will have a simple, yet elegant, interface for viewing Wishlists that have been shared with you as well as viewing Wishlists that you have created- which allows for clarity and the concise presentation of information.

# Customisation of an existing solution

The existing solutions existed above are not able to be customised in a way that are able to fix their inherent issues. For example, you may be able to customise Google Docs’ UI and UX, but the problem of it being a web based online app still exists. I believe that my solution will be the preferred solution, as it will fix a large majority of issues that the existing solutions currently contain. ­­Furthermore, due to the nature of the product development process, there will be chances for user feedback so that my solution can be improved, which would make it far more superior to the current solutions, even if they were to be modified.

# Social and ethical considerations

Wishlists will most likely be handling sensitive user data such as usernames, passwords, emails and the Wishlists themselves:

**Password Storage:**

The software program requires two passwords in order to function properly:

Application password (ensuring that you only get to see Wishlists that have been shared with you)

Email password ( for saving/sharing/loading)

In software applications, passwords should NEVER be stored in plaintext, as users could potentially access the passwords of other users. Passwords should instead be stored in an hashed format. Hashing is a system that turns strings into really long numbers. Example:

“aaa” 🡪 'd6f644b19812e97b5d871658d6d3400ecd4787faeb9b8990c1e7608288664be77257104a58d033bcf1a0e0945ff06468ebe53e2dff36e248424c7273117dac09' ( in hexadecimal notation)

“abcdefg” 🡪 'd716a4188569b68ab1b6dfac178e570114cdf0ea3a1cc0e31486c3e41241bc6a76424e8c37ab26f096fc85ef9886c8cb634187f4fddff645fb099f1ff54c6b8c'

Hashing designed to be irreversible , which ensures that a hash is highly unlikely to be ‘cracked’. When a user logs in the password **entered** is hashed, and compared to the stored “password”(the hashed version of the password used to sign up). If they are a match, then the user is logged in, otherwise the user has entered the wrong password.

Another alternative to keep passwords secure would be hash passwords with a “salt”. A salt is an extra string added onto the end of the password. Salts can be static(the same for every password) or dynamic(changing) For example

If the password entered was “aaa” and the salt is “abcdefg”, the software would hash the string “aaaabcdefg”, which would give a value of

'17ce449456e77d8e4ba9cf3cb1f29b2a15df1eb3f9d3206b92d33c8b3e2c734a88c6ca52bb9629bf319d077ba7a3aa2a3f3ff17a2b5ec8f169bd3f2548fe4568'

Hashing with a salt is very secure, as it adds an extra layer of security to the password storage. There are existing sites/databases which store common passwords and their hashes. The addition of a salt ensures that common passwords ( eg “password”) are changed to (“password” + salt) , which would not be in the table of common passwords

In order for the program to be socially and ethically complicent, passwords in files must be stored in a hashed format. This ensures that if users were to view the users.csv file, they would only see the hashed passwords, not the actual passwords themselves. In terms of password for the email account, that will be stored with use of global variables, which the user will not be able to access.

**Email and Name Storage:**

Whilst Email’s and Names are personal information, they are required for the use of the program, and need to be stored in an easily readable format. This means that these items will not be hashed, and users should acknowledge that there is a chance that other users will be able to view their names and email address. In order to remain socially and ethically moral, this will be clearly outlined in the terms of conditions, so that users are fully aware of this.

**jjj**

# Cost effectiveness

In the development of this solution, there will be no costs as everything used will be of no cost to the developer. These include items such as:

* Software products(free software)
* Laptop(already owned)
* Internet(given to company for free)
* Stationary & Office supplies for diagrams etc(given to company for free)

Hence, cost effectiveness is irrelevant.

# Licensing considerations

The software solution will be released under an open source license, in particular, the GNU GPLv3:

“The GNU GPLv3 is a copyleft license that requires anyone who distributes your code or a derivative work to make the source available under the same terms, and also provides an express grant of patent rights from contributors to users.” <https://choosealicense.com/>

This is due to the fact that the developers wish for anyone to easily access, improve and publish the source code, in order to increase the quality of the problems solved, increase advertising, attract talent and to reduce effort needed to solve a problem. .

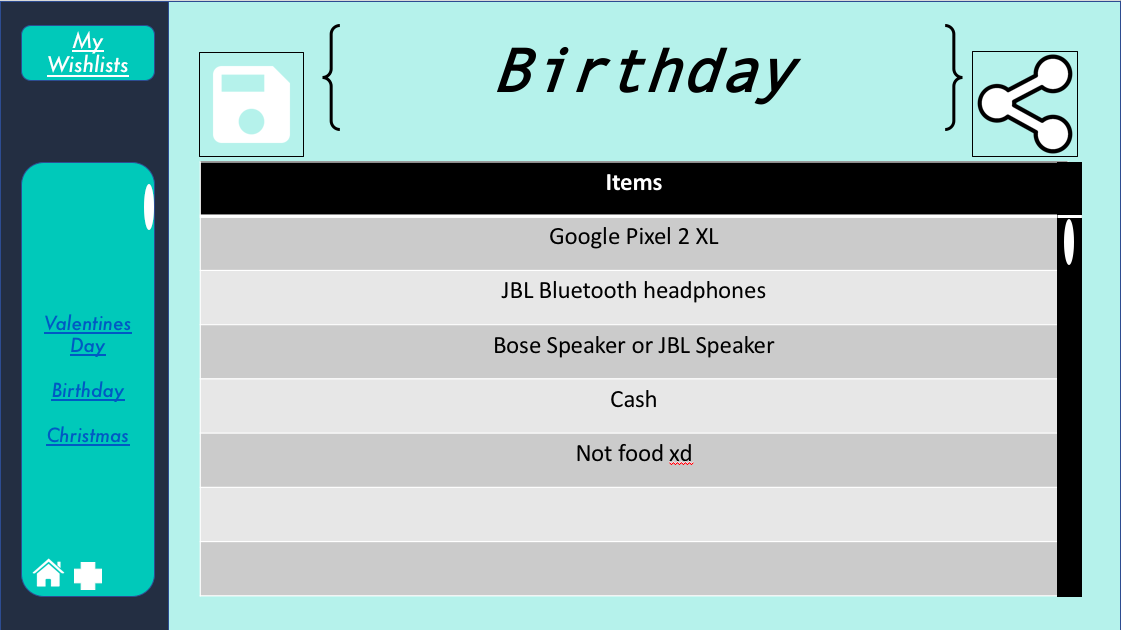
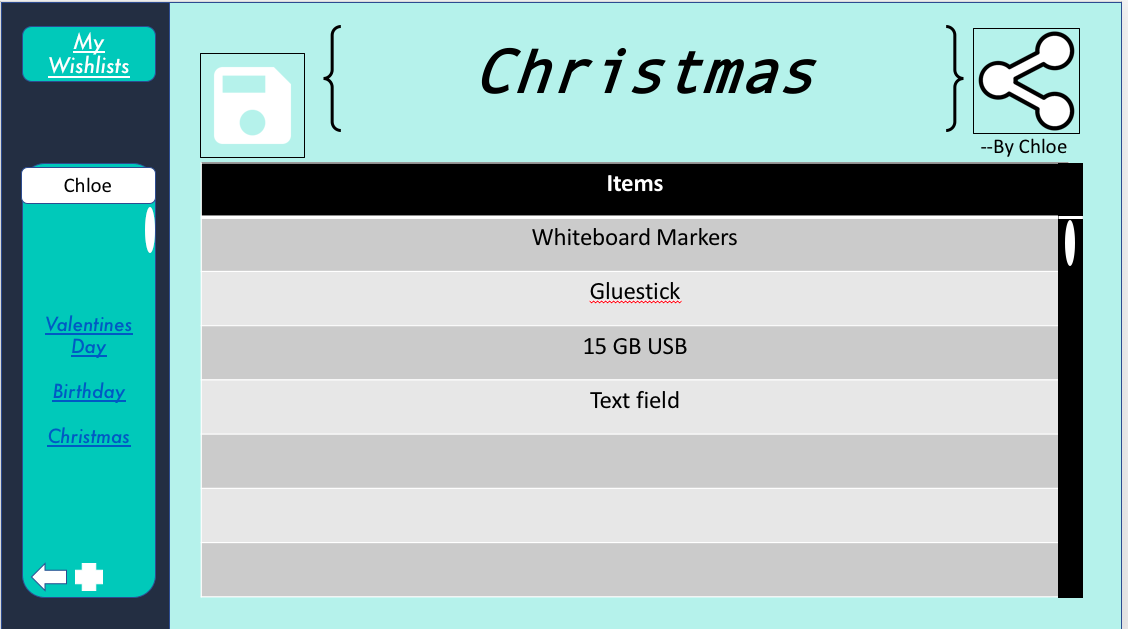
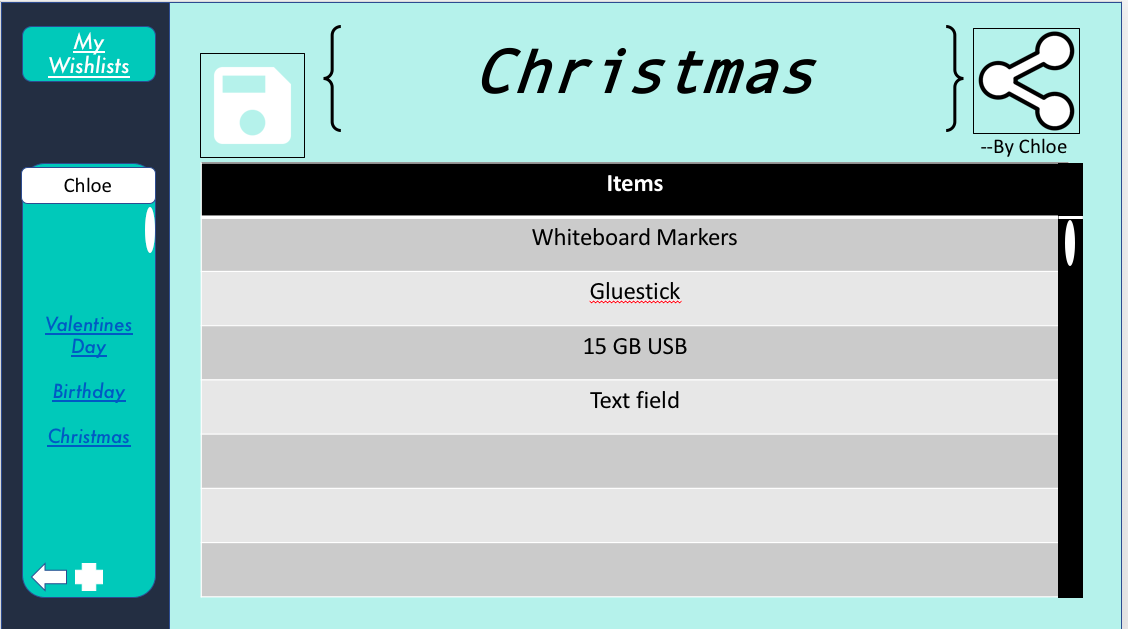
Python freely available

Design Specifications

# Developers perspective

* Data types
  + User (users full name) must be inputted as a string
  + Email must be inputted as a string
  + Password must be inputted as a string
  + Any wishlist item must be inputted as a string
  + Any wishlist title must be inputted as a string
* Data structures
  + The Users.csv file must have data inputted in the following format:
    - user, password, email
  + Wishlists must have data inputted in the following format
    - Item, item, item, item,item,item
    - There may only be 100 items in the file.
* Algorithms
  + All algorithms must be written in Pseudocode
  + All algorithms must demonstrate correct structure and must be capable of solving the problem that the given module is assigned to solve.
  + All algorithms must adhere to the HSC Software Design and Development Course Guidelines
* Variables
  + There will be no restrictions on the style of the variable and function names ( eg camelCase snake\_case kebab-case or PascalCase)
* Software design approach
  + The project will take the structured approach, due to its modular nature which enables the developer to tackle the problem from a logical top-down perspective. This ensures that there is higher quality work and that the program is less likely to contain logic errors
* Quality assurance
  + The quality assurance criteria for the project are as follows:
    - **Efficiency**
      * Efficient algorithms must be used
      * Code is well structured
      * Code is well tested
    - **Integrity**
      * Data validation is thorough
      * There is no ability to access the system illegally
      * Data is stored securely
    - **Reliability**
      * The program must continue to function after time.
      * The system must recover in a timely manner from potential issues
    - **Usability**
      * The use of screen elements, design and colour will remain consistent
    - **Accuracy**
      * The software will perform its functions correctly and according to its specifications
    - **Maintainability**
      * The source code wil be well documented, modular in design, and will include test routines.
    - **Testability**
      * Each submodule will be able to be tested
      * OR
      * Each submodule will be thoroughly tested
    - **Reusability**
      * Submodules will be written with reusability in mind, to ensure that submodules will be able to reused.
* Documentation
  + The following Documentation will be completed as a part of this project:
    - IPO Diagram
    - Storyboard
    - Data Flow Diagram
    - Structure Chart

# Users Perspective

* Interface design
  + Background colour must be: # C2EFE9
  + Foreground colour must be: # 53C3B7
  + Side-bar colour must be: # 242F41
  + There should always be a side-bar with navigation elements on the left hand side as shown in the diagram below
  + 
  + When viewing another user’s wishlist, the user must be aware of the user’s wishlist that they are viewing, their name should be clearly visible in the side bar and under the name of the wishlsit
  + 
  + Also, the “new wishlist” button and the back arrow buton must be visible on all pages where you are able to view another users wishlist
    - 
  + The font for level 1 headings should be Andale Mono
  + The font for level 2 headings should be Futura
  + The font for text(list items) should be Calibri
* Appropriate messages
  + All messages should be clear in their meaning, ensuring that the user understands what message is being conveyed to them.
  + Error messages should be in the hex-colour #FF0000(red) so that it stands out, and should be in the font Futura.
  + Messages should not be too technical, and should be in terms that the user are able to understand.
    - For example a bad error message would look like:
      * “ERROR XOXO2141”, this gives the user no details as to what actually went wrong, and no possible solution.
    - A good error message would look like:
      * “Error! You are not connected to the internet! In order to proceed, please connect to the internet.”, this tells the user exactly what is wrong in plain English and gives them a potential solution.
* Appropriate icons
  + All icons must be white with a black outline and at a relevant size so that they do not crowd the display and match the aesthetics of the program.
* Relevant data formats for display
  + No data besides from text will be displayed, so there is no need for relevant formats.
* Ergonomic issues
  + The most common functions should have a keyboard shortcut to avoid unnecessary strain on the user.
    - The user should be able to create a new wishlist with the keyboard shortcut command N
* Social and ethical issues
  + All personal/private data such as passwords should be stored in an encrypted/hashed format.
  + Users should not be able to see other users’ Wishlists without them sharing it to them

# Gantt Chart

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **Proposed Time** | **Actual Time** | **Not Undertaken** |

| **Section** | **Action/Part** | **TERM 3** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1**  **28/1-3/2** | **2**  **4/2-10/2** | **3**  **11/2-17/2** | **4**  **18/2-24/2** | **5**  **25/2-3/3** | **6**  **4/3-10/3** | **7**  **11/3-**  **17/3** | **8**  **18/3-21/3** |
| **Identifying the problem** | **Identify the problem** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Discussion of relevant issues** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Design Specification** | **From the Developers Perspective** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **From the users perspective** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **System Documentation** | **Create IPO Chart** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Create Data Flow Diagram** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Create Structure Chart** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Create Storyboard** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Create GUI prototype** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **Algorithms** | **Create Algorithms** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| **LogBook** | **Record Entries in Logbook** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |