SDV503 Assessment 3

Patient Health Management system – Caleb Eason

Contents

[1. Project Proposal 2](#_Toc137401434)

[1.1. Overview 2](#_Toc137401435)

[1.2. Timeline 2](#_Toc137401436)

[1.3. Management 2](#_Toc137401437)

[1.4. Proof of Concept 3](#_Toc137401438)

[1.5. Evaluation 3](#_Toc137401439)

[1.6. Conclusion 4](#_Toc137401440)

[2. Design and Implementation 4](#_Toc137401441)

[2.1. Structural Design 4](#_Toc137401442)

[2.1.1. Workflow diagram 4](#_Toc137401443)

[2.1.2. Explanation 5](#_Toc137401444)

[2.2. Data Set 6](#_Toc137401445)

[3. Testing and Debugging 7](#_Toc137401446)

[3.1. Methodology 7](#_Toc137401447)

[3.2. Testing Example 7](#_Toc137401448)

[3.3. Testing Results 9](#_Toc137401449)

[4. Documentation 11](#_Toc137401450)

[4.1. Dependencies 11](#_Toc137401451)

[4.2. Glossary 11](#_Toc137401452)

[4.2.1. Global variables 11](#_Toc137401453)

[4.2.2. Functions 12](#_Toc137401454)

[4.3. Code Comments 13](#_Toc137401455)

[4.4. User Manual 16](#_Toc137401456)

[4.4.1. Starting the program 16](#_Toc137401457)

[4.4.2. Selecting A patient record 16](#_Toc137401458)

[4.4.3. Finding a new record 16](#_Toc137401459)

[4.4.4. Deleting a record 17](#_Toc137401460)

[4.4.5. Modifying a record 17](#_Toc137401461)

[5. References 17](#_Toc137401462)

# Project Proposal

## Overview

This project will deliver a proof of concept for a patient health record management system. The system will provide the local district health board with a tool to access patient information more efficiently, allowing them to provide better care to their patients. The system should enable healthcare providers to access and manage patients’ medical history, diagnoses, medications, test results and other relevant information. The system should also enable patients to view their own information. This proof of concept will fictitious data to simulate patient information.

## Timeline

Here is the proposed timeline for this project. Milestones and due dates are due listed in bold with sub-tasks nested below.

1. **Project Proposal – 06/06/2023**
   1. Overview
   2. Timeline
   3. Management
   4. Proof of concept
   5. Evaluation
   6. Conclusion
2. **Design and Implementation – 11/06/2023**
   1. Structural Design
   2. Data Set
   3. Program
   4. Inputs and outputs
3. **Testing and debugging – 14/06/2023**
4. **Documentation – 17/06/2023**

**Final Due Date – 18/06/2023**

## Management

This project will use the extreme programming software development life cycle methodology. Extreme programming is an agile SDLC methodology that focusses on the technical aspects of software development. Extreme programming is based off five key values: communication, simplicity, feedback, courage, and respect.

Issues and progress can be managed and communicated with GitHub issues. For each stage of the project, a GitHub milestone can be created to organise issues and pull requests. When an issue arises within a milestone, an issue ticket can be created to document and keep track of the issue. Issue tickets can also be used to manage the progress of development tasks. Project members can communicate with each other though the chat function on individual tickets.

## Proof of Concept

This program will allow users to retrieve and modify patient information though a command line interface. When the user asks for a patient’s information, the program will search its stored records to see if there is a record that matches the requested patient. If a match is found, the program will display the information on the requested patient. Otherwise, the program will inform the user that they did not find a marching record and ask them to select a different patient. After the user successfully selects a record, they can enter a command to modify the record, of they can select a different patient. The user can also enter a command to delete a record.

Functional requirements:

* The program should use a text-based user interface to allow the user to interact with the program.
* The user should be able to select a record from the directory to display information on a specific patient.
* The program should provide an error response if a user tries to retrieve information on a patient that is not present in the directory.
* The program should allow the user to modify patient information, but not require modification each time a record is viewed.
* The user should be able to remove a record from the stored information.

## Evaluation

The program will be considered a success if it meets all of the functional requirements outlined in section 1.4. The program can be measured against the following metrics to determine its quality.

* Robustness

The program can correctly handle and respond to a variety of expected and unexpected user inputs. The program handles any errors that arise without crashing.

* Efficiency

The program should use logical operations and call backs in place or repeated code (DRY principles). The program should be optimised to minimise any delay in executing functionality.

* Useability

The program should be easy and intuitive to use for the end user. The program’s UI should always make it clear what the user is able to do at any given point.

* Readability

The code should be written neatly with code comments explaining key decisions and non-intuitive functionality. The code should be interpretable by another developer with minimal effort.

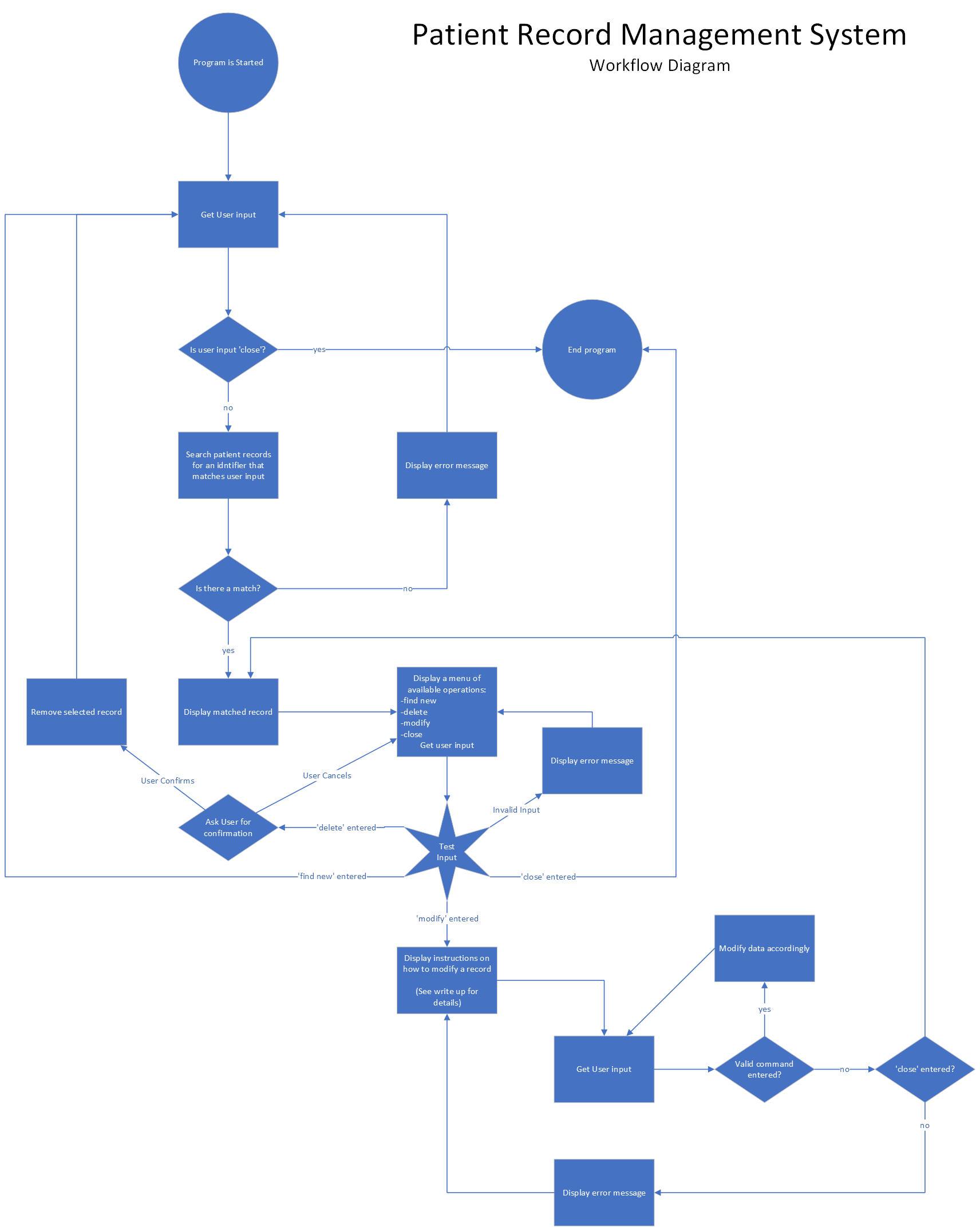
## Conclusion

This system will enable a district health board to improve their operational efficiency by allowing staff to easily retrieve patient information without having to manually search through records. The system will also allow customers to easily access their own health records without having to go through a staff member, saving time for both parties. This proposal provides a simple proof of concept to demonstrate the functionality of the system to the district health board.

# Design and Implementation

## Structural Design

### Workflow diagram



For modifying user records, a command line system will be used. This system will allow the user to select a property of the chosen record and either modify or delete it by entering a single line command.

### Explanation

The program’s structural design follows the DRY philosophy (Don’t Repeat Yourself). Loops and call backs are used, to eliminate any need for code duplication, and a logical, sequential structure is used. For an example of this, see the logic for modifying patient records. This section of the code will loop each time the user enters a command (until ‘close’ is entered). This function will display help text when it is first called, or whenever the user enters an invalid command. If the user enters a valid command however, the help text is not displayed. This provides guidance to users who are not familiar with the system while not unnecessarily repeating information for users who already know what they are doing. When the user is finished modifying a record, instead of calling a new function to display the updated record and ask the user what they want to do next, the program simply calls back to previous logic to accomplish this (Labelled ‘Display matched record’ in the workflow diagram), avoiding repetition of similar functionality. The use of loops and call backs will reduce the amount of code that needs to be written to achieve the programs functionality. This will make building the program less intensive and easier to execute (In comparison to a program that does not follow DRY principles). Maintaining the program will be more intuitive as new developers will have less code to familiarise themselves with. The program’s logical, sequential structure should be easy for developers to follow and implement.

## Data Set

Proposed data structure: Array of objects.

Data to include for each record:

* Patient number
* Full name
* Date of Birth
* General practitioner
* Appointments

Example:

patients = [  
{  
patientNumber: 0  
fullName: ‘John smith’,  
dateOfBirth: ‘11/02/1991’  
generalPractitioner: ‘Sally West’  
appointments: [‘knee surgery on the 12/08/2023’,’general checkup 26/11/2023’]  
},  
{  
patientNumber: 1  
fullName: ‘Kate brown,  
dateOfBirth: ‘02/04/2005’  
generalPractitioner: ‘Charlie Chut  
appointments: [‘general check-up on 18/09/2023’]  
},  
etc..  
]

# Testing and Debugging

## Methodology

The program was tested throughout development to ensure robust and accurate functionality in line with agile/XP principles. For debugging and testing console.log() was used to display the results of hidden procedures and determine exactly what code was and wasn’t being executed.

A screenshot of a computer code

Description automatically generated with low confidence

After the program was finished, comprehensive testing was applied to all facets of the program. Every user input was tested with expected, invalid and boundary inputs to ensure the program could handle any and all user inputs.

## Testing Example

Here is an example of the testing methodology applied.

|  |  |  |  |
| --- | --- | --- | --- |
| Item to test: Command line input for modifying record information | | | |
| Input | Expected Result | Actual Result | Pass/Fail |
| modify fullName “Clara Ruth”  (expected input) | The program should set the value of fullName to “Clara Ruth” |  | Pass |
| modify dateOfBirth “01/05/2000”  (expected input) | The program should set the value of dateOfBirth to “01/05/200” |  | Pass |
| modify generalPractitioner “Dean Stuart”  (expected input) | The program should set the value of generalPractitioner to “Dean Stuart” |  | Pass |
| modify Name “Clara Ruth”  (invalid input) | The program should reject the property “Name” as it does not appear in the record |  | Pass |
| modify patientNumber “5”  (invalid input) | The program should not allow modification of patientNumber |  | Pass |
| modify appointments “surgery on 31/11/2024”  (invalid input) | The program should not allow modification of appointments, and direct the user to the append command |  | Pass |
| modify dateOfBirth  (invalid input) | The program should reject the input as no new value was specified | \*The program functioned correctly but the formatting of the error message was slightly off, a quick fix remedied this: | Pass\* |
| append appointments “Surgery on the 04/05/2023”  (expected input) | The program should add “Surgery on the 04/05/2023” to the array of appointments |  | Pass |
| append checkups “Surgery on the 04/05/2023”  (expected input) | The program should reject the property “checkups” as it does not appear in the record |  | Pass |
| append fullName “Clara Ruth”  append dateOfBirth “01/05/2000”  append generalPractitioner “Dean Stuart”  append patientNumber “5”  (similar invalid inputs) | Each input should produce the same output:  The input should be rejected as append can only be performed on properties with array values | (Each input produces the same output) | Pass |
| append appointments  (invalid input) | The program should reject the input as no new value was specified |  | Pass |
| change fullName “joe”  (invalid input) | The program should reject the unrecognized command |  | Pass |
| MODifY fullName “John Hill”  (expected boundary input) | The program should ignore the capitalisation of the modify command and set the property of fullname to “john Hill” |  | Pass |
| Modify FULLNAME “Cal Riveir”  (invalid boundary input) | Properties are case sensitive so “FULLNAME” should be rejected |  | Pass |
| null  (no value entered)  (invalid input) | The program should reject the empty input as an unrecognised command | Program Crashed | Fail |
| close  (expected input) | The program should exit the modification function and return to the record operations function, displaying the modified record and awaiting input. |  | Pass |

## Testing Results

Because thorough testing was carried out during program construction, final testing revealed only 2 bugs/errors.

In the record modification command line, entering an undefined value would cause the program to crash (See example testing). This was fixed by adding a case for undefined values at the start of the if/else chain. If an undefined value is entered, all other cases are not tested to prevent a crash.



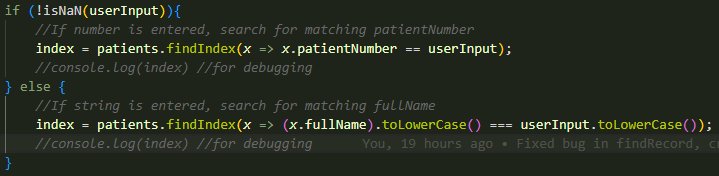
Now entering an undefined value will simply cause the program to skip to the next iteration of the while loop and request user input again.



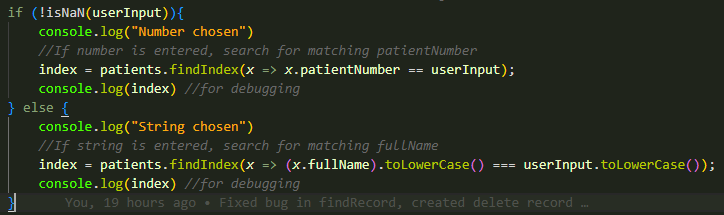
Another bug occurred when selecting a patient record. Entering an empty value would cause the system to select patientNumber 0 (if present in the patients list. If there was no patientNumber 0, the input would be rejected as expected).

This was theorised to be an issue with the index assignment system in the findRecord() function.

Index assignment code block:



To test what was going wrong, console.log() functions were added into the if/else statement to see which was being called on an empty value, and what the index was being assigned to.



This revealed when an empty value is entered, the first condition is run, and the index is assigned to 0.

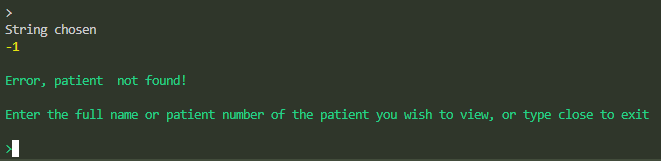


Now that the cause of the issue is identified, a fix can be implemented.

To fix this problem, another condition was added to the first if statement to test if the length of the inputted value was greater than 0.



Testing the program with the new condition produced the following output:



The program successfully rejected the empty input. The debugging code revealed that the program was selecting else statement, which did not produce a match against the empty string (signified by the index being -1).

Note, debugging code has been removed from the main release, but is present in the dev branch.

# Documentation

## Dependencies

This program is dependant on the readline-sync module to retrieve user input.

The module has been included in the node\_modules file.  To install, use 'npm install'.  If this doesn't work visit 'https://www.npmjs.com/package/readline-sync' to install manually

## Glossary

### Global variables

Commands

Key words used in input validation to execute certain behaviour. Used in place of hardcoded values to make modifying the program easier. All use string datatype.

* exitFunctionCMD - Close the program or exists the current function
* deleteRecordCMD - Calls deleteRecord() when used inside recordOperations()
* modifyRecordCMD - Calls modifyRecord() when used inside recordOperations()
* findRecordCMD - Returns to main, then calls findRecord() when used inside recordOperations()
* modifyPropertyCMD - Triggers the modify command when used in modifyRecord()
* appendToPropertyCMD - Triggers the append command when used in modifyRecord()

Text outputs

Used in place of long, hard coded text outputs to make reading functions easier. All use string datatype.

* welcomeMessage - The message displayed when first opening the program
* OperationInstructions - The command list that appears when recordOperations() is called
* ModificationInstructions - The instructions displayed when modifyRecord() is called

General

* patients - Array containing patient information. Each patient is represented by an individual object containing the following properties:
  + patientNumber - A unique identifier (number)
  + fullName - Patients full name (string)
  + dateOfBirth - Patients birth date (string)
  + generalPractitioner - Patient’s preferred GP (string)
  + appointments - An array of strings detailing scheduled appointments for the patient

### Functions

main()

The root function called upon program start, displays a welcome message then runs the main functional loop of the program.

Local variables:

* index - stores the index in the patients list of the selected patient record, also used to end the program if set to the value of exitFunctionCMD
* status - Used to end the program if set to the value of exitFunctionCMD

findRecord()

Takes user input to select a record from the patients array and returns the result. Can select a record based on patientNumber or fullName.

Local variables:

* userInput - Stores user input for processing

Parameters:

* index - parsed from main()

recordOperations()

Displays the selected record and a list of available user actions (find a new record, delete record, modify record, end program). Takes user input to select an action, then either returns to main or executes another function.

Local variables:

* userInput - Stores user input for processing

Parameters:

* index - parsed from main()

deleteRecord()

Deletes the selected record, first asking for user confirmation. Returns to main if record is deleted, recalls recordOperations() if not.

Local variables:

* userInput - Stores user input for processing

Parameters:

* index - parsed from recordOperations()

modifyRecord()

Runs a command-line like interface, enabling the user to enter commands to modify properties of the selected record.

Local variables:

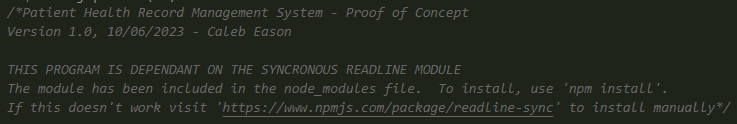
* userInput - Stores user input for processing
* command - stores user input, separated into an array of individual words

Parameters:

* index - parsed from recordOperations ()

## Code Comments

Here are screenshots of some of the code comments in the program.



A screen shot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font

Description automatically generated

A picture containing text, screenshot, font, software

Description automatically generated

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer code

Description automatically generated with low confidence

A picture containing text, font, screenshot

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated

A picture containing text, screenshot, font

Description automatically generated





A screen shot of a computer program

Description automatically generated with low confidence

## User Manual

### Starting the program

To begin the program, make sure the terminal is set to the correct directory, then use the command :

node src/app.js

A welcoming message will appear, press enter to move on to selecting a patient record.

### Selecting A patient record

To select a patient’s health record, you can enter either the patient’s full name, or their patient ID number. For example, to retrieve the patient information of “John Smith”, patient ID number 0, you can enter either ‘john smith’ or ‘0’. If you enter a name or ID number that is not present in the system, the program will inform you the entered patient is not present, and you will have the opportunity to reselect a patient. You can also type ‘close’ to exit the program.

Once you have selected a patient record, their information will be displayed along with a list of available actions. You can find a new patient record, delete the selected record, modify the selected record, or you can close the program by entering ‘close’.

### Finding a new record

To look up a record, enter ‘find-new’ this will allow you to select a new record as described in section 4.3.2.

### Deleting a record

To delete the selected record, enter ‘delete-record’. You will be asked for confirmation before the record is deleted. To confirm the deletion, enter ‘DELETE’ (case sensitive), to cancel the deletion, enter any other input or simple press ENTER.

### Modifying a record

To modify a property, enter ‘modify-record’. This will open the record modification menu. From here you can modify the properties of the selected record by entering command inputs.

To modify a property of the selected record, the following command can be used:

modify <property name> “<new value>”

<property name> refers to the property in the record to be modified, such as fullName or dateOfBirth. Properties are case sensitive, so entering the property “FULLNAME” will cause an error.

<new value> refers to the value the selected property will be changed to.

For example, to change the patients full name to john smith you can use the following command:

modify fullName “john smith”

Note: You can not change a patient’s ID number.

To append a new value to a property containing a list of values, use:

append <property name> “<new value>”

Use this command to add new appointments for the selected patient.

For example, to add the appointment ‘check up, 11/07/2024’ you can use the following command:

append appointments “check-up, 11/07/2024”

Note: You can only use this command on properties that can contain multiple values.

To exit and save changes, use:

close

# References

Nimble. (2023, March 4). What is extreme programming (XP)? - Values, principles, and practices. <https://www.nimblework.com/agile/extreme-programming-xp/>

Below reference refers to code used within the program:

Séguret, D. (2013, September 9). Split string into words with whitespace unless in between a pair of double quotation marks. Stack Overflow. <https://stackoverflow.com/questions/18703669/split-string-into-words-with-whitespace-unless-in-between-a-pair-of-double-quota>