|  |
| --- |
|  |
| Home Link |
| Methods in Software Engineering |
| Project Book |
| **Authors:**  Tslil Aharon - 318320694  Linoy Noe - 316187863  Noam Bassous – 206843237  Date: |
| **‏** |

|  |
| --- |
|  |

Abstract

Following the war, a significant number of people found themselves without shelter, forced to look for temporary accommodation in hotels or various guesthouses. Entire families with children were crammed into small rooms, which created difficulties in managing the daily routine. Finding temporary housing can be a challenging task, and our idea is to successfully deal with this dilemma.

Our platform will connect apartment owners who are willing to donate their properties to evacuees and people who remain homeless, looking for short-term rentals. The system will include a database of available properties, allowing filtering and sorting based on specific parameters.

Each property listing will provide detailed information about nearby amenities such as grocery stores, supermarkets, playgrounds and medical facilities. The system aims to encourage property owners to donate temporary housing through the creation of a competitive rating mechanism that strengthens and promotes social ties and contribution to the community. Feedback and ratings can be provided at the end of each hospitality experience, fostering a sense of community and gratitude. Through this innovative approach, our platform seeks to alleviate the housing crisis and enable a supportive environment for those in need. In the system it will be possible to receive offers according to the personal preferences of the evacuee and it will be possible to rate the owner of the property after the stay according to the hospitality experience.

Table of Contents

[1 INTRODUCTION 6](#_Toc159048112)

[1.1 Problem Description 6](#_Toc159048113)

[1.2 HomeLink Motivation 7](#_Toc159048114)

[1.3 HomeLink Goals 8](#_Toc159048115)

[1.4 Overview of the HomeLink Approach 8](#_Toc159048116)

[1.5 Usage scenarios 8](#_Toc159048117)

[1.5.1 Scenario 1 - Subsets of existing files/directories 8](#_Toc159048118)

[1.5.2 Scenario 2 - Using the same aggregation into a different order of folders 8](#_Toc159048119)

[1.5.3 Scenario 3 - Using different aggregation into new folders 9](#_Toc159048120)

[1.6 HomeLink audience 9](#_Toc159048121)

[1.7 Glossary 9](#_Toc159048122)

[1.7.1 Generic terminology 9](#_Toc159048123)

[1.7.2 HomeLink specific terminology 10](#_Toc159048124)

[2 TECHNOLOGICAL SURVEY 12](#_Toc159048125)

[3 REQUIREMENTS AND SPECIFICATION 13](#_Toc159048126)

[3.1 HomeLink functional requirements 13](#_Toc159048127)

[3.2 HomeLink non-Functional requirements 13](#_Toc159048128)

[3.3 Specification - the scope of the HomeLink project 14](#_Toc159048129)

[3.4 The HomeLink approach 14](#_Toc159048130)

[4 THE ARCHITECTURE 16](#_Toc159048131)

[4.1 HomeLink Data 16](#_Toc159048132)

[4.1.1 The internal and external representations (of files and folders) 16](#_Toc159048133)

[4.1.2 Initial start-up data 16](#_Toc159048134)

[4.2 HomeLink Processes 17](#_Toc159048135)

[4.3 Roles 18](#_Toc159048136)

[4.4 The life-cycle view of a system 18](#_Toc159048137)

[5 SYSTEM DESIGN 21](#_Toc159048138)

[5.1 Data components 21](#_Toc159048139)

[5.1.1 The Internal Representation (InRep) data structures 21](#_Toc159048140)

[21](#_Toc159048141)

[21](#_Toc159048142)

[5.1.2 The External Representation (ExRep) display 22](#_Toc159048143)

[5.2 Process components 22](#_Toc159048144)

[5.3 Communication components (user-computer interaction) 23](#_Toc159048145)

[5.4 Interactions 23](#_Toc159048146)

[5.4.1 State diagram 23](#_Toc159048147)

[5.4.2 Design sequence diagram 25](#_Toc159048148)

[5.5 System Architecture 25](#_Toc159048149)

[6 IMPLEMENTATION 27](#_Toc159048150)

[6.1 Algorithm Design 27](#_Toc159048151)

[6.2 Interfaces 27](#_Toc159048152)

[6.3 Development environment 31](#_Toc159048153)

[6.4 Programming languages 32](#_Toc159048154)

[6.5 Risk Management 32](#_Toc159048155)

[6.6 Exceptions Management 32](#_Toc159048156)

[6.7 Versions Control 32](#_Toc159048157)

[6.8 Project Management 32](#_Toc159048158)

[6.9 Code 33](#_Toc159048159)

[7 SYSTEM VALIDATION 34](#_Toc159048160)

[8 SUMMARY, EVALUATION, CONCLUSIONS AND FUTURE WORK 35](#_Toc159048161)

[8.1 Summary 35](#_Toc159048162)

[8.2 Evaluation and conclusions 35](#_Toc159048163)

[8.3 Future work 35](#_Toc159048164)

[9 REFERENCES 37](#_Toc159048165)

[10 APPENDIX A: Background material 38](#_Toc159048166)

**Table of Figures**

[Figure ‎1‑1: A File Browser (FB) or File Manager is a tool for viewing and managing the File System. 6](#_Toc488324469)

[Figure ‎1‑2: The default view of the File System (FS) as represented by standard File Browsers. The file browser's view reflects the contents and organisation of the file system and any non-trivial changes applied in the context of the view affect the file system. 7](#_Toc488324470)

[Figure ‎1‑3: Alternative points of view of the File System can be created and modified - without affecting the contents and structure of the File System 7](#_Toc488324471)

[Figure ‎1‑4: Scenario 1 - Subsets of existing files/directories can be put inside a folder 8](#_Toc488324472)

[Figure ‎1‑5: Scenario 2 - Using the same file aggregation into a different hierarchy of folders 9](#_Toc488324473)

[Figure ‎1‑6: Scenario 3 - Using different file aggregation into new folders 9](#_Toc488324474)

[Figure ‎1‑7: A *Virtual View* consists of an *Internal Representation* & *External Representation*. 11](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324475)

[Figure ‎2‑1: Q-Dir application screen shot 13](#_Toc488324476)

[Figure ‎3‑1: The ProFiler approach – the initial default state of the FS is used to create alternative Views using the provided Transformations carried out by the user. Saving a view and retrieving it later is also possible 16](#_Toc488324477)

[Figure ‎4‑1: A view consists of the Internal and External Representations. The ExRep is the external display of the InRep in a graphical form that can be manipulated by the user 17](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324478)

[Figure ‎4‑2: The ProFiler starting point is the extraction of the data from the File System (FS) into the default Internal Representation. The user can transform the internal representation to create alternative virtual views 17](#_Toc488324479)

[Figure ‎4‑3: A user can save a view by storing its Internal Representation in the InRep Repository together with additional data. It can be retrieved from the InRep Repository to continue to work on it 18](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324480)

[Figure ‎4‑4: Transformations - the user can modify or create a new view from one or more existing views by activating the given transformations 19](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324481)

[Figure ‎4‑5: The initial phases of the ProFiler application life-cycle where the FS contents are extracted as the starting point for creating virtual views of the file space 20](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324482)

[Figure ‎4‑6: At any point in the creation of a virtual view it is possible to save a view for later use 20](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324483)

[Figure ‎4‑7: The latter phases of the ProFiler application life-cycle – retrieval of a previously saved view and the subsequent transformations on it 21](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324484)

[Figure ‎5‑1: Tree, File and Folder class diagrams – components of the Internal Representation (InRep) 22](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324485)

[Figure ‎5‑2: The InRep List contains the internal representations of the created views 23](#_Toc488324486)

[Figure ‎5‑3: The display of ProFiler is made of the external representation of the currently available views 23](file:///C:\Users\Home\Documents\A%20a%20Student%20projects%20Admin\A%20Proj%20Docs\5%20Project%20Thesis\A%20Latest%20to%20students\A%20Thesis%20example%20yh.1.docx#_Toc488324487)

[Figure ‎5‑4: The overall design of ProFiler showing the components and the interactions among them - carrying out the processes, using the data and supporting the roles defined in the architecture 24](#_Toc488324488)

[Figure ‎5‑5: ProFiler state diagram 24](#_Toc488324489)

[Figure ‎5‑6: ProFiler sequence diagram 26](#_Toc488324490)

[Figure ‎6‑1: The ProFiler system configuration - the result of mapping the ProFiler design (as shown in Figure 5.5) onto the 'Three tier software architecture' and the related components 28](#_Toc488324491)

[Figure ‎6‑2: ProFiler activity sequence diagram 29](#_Toc488324492)

[‎6‑3: ProFiler layered system configuration with the libraries they use 30](#_Toc488324493)

[Figure ‎6‑4: Default view (left side) 31](#_Toc488324494)

[Figure ‎6‑5: Guide button 31](#_Toc488324495)

[Figure ‎6‑6: ProFiler sections 32](#_Toc488324496)

[Figure ‎6‑7: Drag & Drop feature 32](#_Toc488324497)

[Figure ‎6‑8: ‘Save tree’ button and ‘open and choose tree’ button 33](#_Toc488324498)

[Figure ‎6‑9: Search/filter files and folders 33](#_Toc488324499)

[Figure ‎6‑10: Working station 33](#_Toc488324500)

[Figure ‎6‑11: Saving a tree 34](#_Toc488324501)

[Figure ‎6‑12: Restore saved tree 34](#_Toc488324502)

[Figure ‎6‑13: Restored tree 34](#_Toc488324503)

[Figure ‎7‑1: Saving transformation and being able to apply them later semi-automatically 37](#_Toc488324504)

[Figure ‎7‑2: ProFiler views should be made aware of changes made to the File System - keeping things consistent may prove tricky 37](#_Toc488324505)

[Figure ‎7‑3: Allowing specified modifications performed in ProFiler to affect the File System – an open question 37](#_Toc488324506)

# INTRODUCTION

A significant part of our day is spent in front of a computer, managing our documents and our information space. We store, file, organise and retrieve information from a variety of document sources in our computer and outside it. The growth of the importance of the data in our world causes more and more documents to flow into our computer every day. Tools which visualise, maintain and manage our documents are therefore in greater need than ever. Different approaches and tools that allow users to manage their own documents are the concern of the Personal Information Management (PIM) field [PIM 15].

This section should also include practical and/or theoretical implications of the project.

## Problem Description

## ProFiler Motivation

## ProFiler Goals

## Overview of the ProFiler Approach

## Usage scenarios

### Scenario 1 - Subsets of existing files/directories

### Scenario 2 - Using the same aggregation into a different order of folders

### Scenario 3 - Using different aggregation into new folders

## ProFiler audience

## Glossary

The following terms are divided into two categories: generic and ProFiler specific terms.

### Generic terminology

* **Personal Information Management** (**PIM**): a field of work which refers to the study and practice of the activities people perform in order to acquire, organize, maintain, retrieve and use personal information items such as documents (paper-based and digital), web pages and email messages for everyday use to complete tasks (work-related or not) and fulfil a person’s various roles (as parent, employee, friend, member of community, etc.) [PIM 15].
* **File System (FS)**: a repository of information containing information about the files and directories in our computer. It is used to create, manipulate and delete files and directories. The FS is maintained and manipulated by the operating system and any access.
* **File**: a basic resource for storing information in a computer that is usually based on some kind of durable [storage](http://en.wikipedia.org/wiki/Computer_storage). Files are sometimes referred to as **Documents**.
* **Folder/Directory**: a [file system](http://en.wikipedia.org/wiki/File_system) cataloguing structure which contains (references to other[) files](http://en.wikipedia.org/wiki/Computer_file) and possibly other folders/directories.
* A **File Browser** (**FB**) or **File Manager** (Figure 1-1): a computer program which provides a user interface for managing **Files** and **Folders** that reside in the **File System**. The most common operations performed on files or groups of files include: viewing or manipulating (creating, opening, renaming, modifying attributes/properties, moving, copying, deleting and searching) the files and folders [File Manager 15].
* **File attribute:** usually in the form of 'name – value' pair that can be used to characterise a document beyond the default attributes that every file has such as name, date of last change etc.
* **Graphical User Interface (GUI):** is a type of interface that exploits the graphics capabilities of a computer to enable users to interact with applications through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.
* **Drag & Drop:**  is a sequence of gestures in GUIs where by using a pointing device (e.g. mouse) the user "grabs" a virtual object (e.g. icon) by pointing and clicking on it and "dragging" it to a different location or onto another virtual object. The object being dragged, the dragging and the location or object to which it is dragged are translated into an "object" "acts" on "subject" phrase that is interpreted by the application of which the GUI acts as a front-end.
* **Web application** or **web-app** is a client-server software application in which the client (or user interface) runs in a web browser and the server is usually run on a remote server (but can also run locally). A **stand-alone application** runs stand alone in a desktop or laptop computer without requiring a web browser or a server.

### ProFiler specific terminology

* **Transformation**: the process of changing a specific File/folder configuration into another by using one of the following processes: The transformations act on the Internal Representations, resulting in a change to the External Representation, A view is called virtual since it does not affect the FS. The transformations are:
  + **Aggregation**: selecting and combining files\folders into other folders. Using similar\partial\different aggregations of the original structure.
  + **Filtering**: creating subsets of the files and folders according to a specified search.
  + **Re-Structuring**: building a new organization of the files and folders.
* **Virtual view, internal and external representations** (Figure 1-7):
  + **Internal Representation (InRep)**: the way in which ProFiler stores the contents and structure of the files internally (as a graph\tree) for its internal use.
  + **External Representation** **(ExRep)**: the graphical displaying of the ProFiler Internal Representations to the user. The external representation displayed by a GUI will enable the user to see **👁** the current virtual view and to manipulate it by using the 'drag&drop' mode of interaction **☞** .
  + **Virtual view**: the result of displaying the Internal Representation to the user as the External Representation. It can be regarded as a combination of the internal and external representation. The ProFiler views are called virtual because no matter what changes are applied to them – it does not modify the content and structure of the File System. However – throughout this document – when we refer to ProFiler virtual views - we use the term view alone. Where we want to distinguish ProFiler's views from other File Browser views – we use the term virtual views.
  + **FS default view**: the starting point for the ProFiler is the extraction of the current state of the files and folders in the File System. This state is called the FS default view.

|  |
| --- |
| Figure ‎1‑7: A *Virtual View* consists of an *Internal Representation* & *External Representation*. |

# TECHNOLOGICAL SURVEY

* הכוונה להשוואה בין שפות תכנות, מסדי נתונים, אלגוריתמים, שירותי ענן וכן הלאה **שרלוונטיים לאילוצים**, **לדרישות** **ולצרכי** הפרויקט **הספציפי** שלכם.
* סעיף זה מוגבל לשני עמודים, לכל היותר.

# REQUIREMENTS AND SPECIFICATION

**Functional** requirements of the project are the necessary conditions that relate to the functionality of the system. **Non-functional requirements** of the project are requirements which do not relate to functionality but to aspects that influence the manner in which the functionality is provided.

## ProFiler functional requirements

The following requirements relate to the basic functionality of ProFiler:

1. **Multiple alternative views**: the user will be able to create different virtual views of the contents of the FS - the provision of alternative views that differ from the basic view of the file service by being able to filter, aggregate and structure the files and folders as needed. It should be possible to construct such views in accordance with the needs of the user.
2. **Starting point**: the starting point for the alternative views is the initial state of the FS (also referred to as the default FS view). The application will extract the FS information about the files and folders in the system and use it as a starting point for creating different views.
3. **Basic Transformation operations**: provide a set of transformations that can manipulate views without causing any changes to the file system itself (Figure 1-3 & 4-1) thereby creating different views of the files and folders. The manipulations are enabled by a number of basic transformations that are provided by the application and that the user can exploit: Filtering, Aggregating & Restructuring.

Each view that is produced can be the starting point for additional transformations.

1. **Display and manageability of views**: the transformations are displayed graphically to the user. The displayed views can be further manipulated to change the view or create more views. The manipulations include the ability to add/modify/delete/rename files or directories and the change the structure of the folders.
2. **Separation of views from FS**: the creation and manipulation of the original FS information and the subsequent views must not affect the original information of the FS.
3. **Persistence of views**: the ability to make the views persistent by being able to store and retrieve them for later use- a view can be named and saved for later retrieval. The retrieved view can be further manipulated as necessary. Unsaved views will not be persistent – closing the application will result in losing them unless they have been saved.
4. **Acting on files**: the application will be able to open the files in the views according to the type of applications that can process them (e.g. Word to open word documents, PowerPoint to open PowerPoint documents, etc.).
5. **FS changes & consistency checks**: the ability to update the alternative view (or at least warn the user) when changes to the FS take place. Such changes are: insertion, deletion, name changes, moving of a file/folder.
6. **ProFiler changes:** certain changes whichare made by the user through ProFiler should be reflected back to the FS. For example, a change of the name of a file/folder.
7. **Transformation automation**: the semi-automation of the process of creating the views so that when changes take place in the file system, a resulting non-consistent view can be made consistent without too many difficulties.

## ProFiler non-Functional requirements

The following is a list of non-functional requirements:

1. **Fail-safe:** the manipulations of the original FS data and subsequent views will under no circumstances affect the FS.
2. **Usability**:

* **Graphical manipulation**: the manipulations of the views will be done via a **graphical** **user interface**. A graphical user interface will provide the user with a view of the files and directories (the external representations) and will supply the transformations as operations that can be performed on this representation.
* **Help**: user tutorial and help facilities will be part of the application.

1. **Portability**: the application should be runnable on different platform without having to make any or just minor alterations.
2. **Maintainability**: the software should be constructed in a modular fashion with clear interfaces to ease the problem of maintenance and extensions.
3. **Installation** **ease**: the software should be easily installed on different platforms. This may necessitate the addition of an Installation package.

## Specification - the scope of the ProFiler project

After a detailed consideration of the requirements in the context of the suggested approach, and taking into account the resources available, it was decided that:

* **Inside the scope of the project**: Functional requirements 1-8 and all the non-functional requirements will be addressed by the approach described above.
* **Outside the scope of the project**:

1. **FS changes & consistency checks**: any changes in the FS caused by the user (through the use of the default FS browsers) which affect the views derived by ProFiler should be conveyed to the user. The ProFiler will not be notified of changes in the FS and the changes made in ProFiler will have no effect on the FS.
2. **ProFiler changes**: Certain changes which are made by the user through ProFiler should be reflected back to the FS. for example, a change to the name of a file/folder.
3. **Transformation automation**: the semi-automation of the process of creating the views so that when changes take place in the file system, a resulting non-consistent view can be made consistent without too many difficulties.

## The ProFiler approach

To satisfy both the functional and non-functional requirements stated in the previous section, the following approach to the development of the application was adopted (Figure 3-1):

* The application will start by presenting the current state of the FS as the initial default view.
* The creation and modification of the alternative views will be carried out manually – special labelling or meta-file data will not be used to create the views. This will also include the creation of new folders and folder hierarchies.
* At any point, the user will be able to create a new view and combine the information from previous views into the new view and modify it as necessary. The following transformation operations will be available to the user:
  + Copying parts of the default view or the current view.
  + Changing the structure of the folders hierarchy.
  + Adding new Folders.
  + Renaming existing files/folders in the new view.
  + Using auxiliary functions such as filtering to help select and aggregate files and folders.
* The external representation of the view will be displayed graphically to the user. The user will be able to use the 'Drag&Drop' mode of interaction in order to specify the transformations that are required in order to produce a new view or to modify a current view.
* The user will be able to delete any files or folders from a view. The user will also be able to delete a view if it is no longer needed.
* The user can decide to make a specific view persistent by storing it with annotations. The stored view can be retrieved at a later stage and the work on it can be resumed.
* In the ProFiler minimal configuration - none of the above actions should have any effect on the FS.

|  |
| --- |
|  |

Figure ‎3‑1: The ProFiler approach – the initial default state of the FS is used to create alternative Views using the provided Transformations carried out by the user. Saving a view and retrieving it later is also possible

# THE ARCHITECTURE

In order to describe the architecture of the ProFiler application we start by describing the data that is needed, where it is originates and how it is transformed. This is followed by explaining the roles that are involved with the application. Finally, the activities that take place when using the system are explained resulting in a description of the system life cycle – the stages through which the system goes when it is being used.

## ProFiler Data

### The internal and external representations (of files and folders)

The basic data unit of ProFiler consists of a View (Figure 4-1). The view comprises of two things which are kept separate from each other:

* **Internal representation (InRep) –** this is the way in which ProFiler stores the contents and structure of the files/folders for its own use. This will be in the form of a tree data structure (see the Design section for further details).
* **External representation (ExRep)** **–** this isthe graphical display of the ProFiler internal structure. The user can specify the transformations to be applied to the internal representation by manipulating the external view – using the Drag&Drop GUI technique.

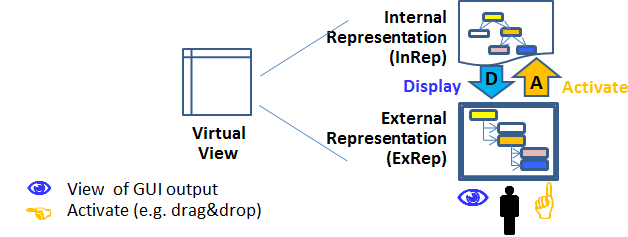


Figure ‎4‑1: A view consists of the Internal and External Representations. The ExRep is the external display of the InRep in a graphical form that can be manipulated by the user

### Initial start-up data

The basic view of ProFiler is created from the initial File System (FS) data (Figure 4-2) and is identical to the view that other standard file browsers provide.

|  |
| --- |
| Figure ‎4‑2: The ProFiler starting point is the extraction of the data from the File System (FS) into the default Internal Representation. The user can transform the internal representation to create alternative virtual views |

The ProFiler requests the File System’s file and folder structure from the Operating System and stores it in its default Internal Representation.

#### View persistence

A user can save any of the created views at any point in time by storing the Internal Representations of the view (Figure 4-3). The user can retrieve it for later use (Figure 4-4) and further manipulate it. The data will be stored in a special **InRep Repository** (Figure 4-4, 4-5). The saved representation is named by the user, and includes date, time and any text annotations the user wishes to add.

|  |
| --- |
| Figure ‎4‑3: A user can save a view by storing its Internal Representation in the InRep Repository together with additional data. It can be retrieved from the InRep Repository to continue to work on it |

## ProFiler Processes

The processes provided by ProFiler are the transformations which provide the user the ability to create, maintain and manage different views of the files and directories. The processes include:

1. **View transformations** (Figure 4-4):
2. **Filter (search)/select** - using selection criteria (file type, date etc.) a user can use filtering to make a smart search. Such criteria can also be certain folders that contain the files\folders. The user can also apply multiple searches – an additional search on the previous search result (Figure 1-5).
3. **Aggregate** - aggregate files\folders into new virtual folders. This can be done in several ways: using similar\partial\different aggregations as the original structure, using similar\different order of the original parent folders and others (Figure 1-6).
4. **Re-Structure** - creating alternative structures of the files and folders by building a new organization of their structure (Figure 1-7).
5. **View display and Actions** (Figure 4-4):

* **View display** (output): displaying the Internal Representation in a graphical manner to the user. ProFiler is responsible to display this view after each change (transformation) that the user applies.
* **View actions** (input): receiving actions in a Drag&Drop manner from the user. ProFiler takes these actions and activates the relevant transformation to the current view.

1. **Internal Representation saving and retrieving**:

* The process which saves the internal representation selected by the user allows the user to add annotations to it as necessary (Figure 4-6).
* The retrieval process allows the user to select from the previously saved internal representations (Figure 4-7).

|  |
| --- |
| Figure ‎4‑4: Transformations - the user can modify or create a new view from one or more existing views by activating the given transformations |

## Roles

The ProFiler project has one main role associated with it – the user who serves as the end-user as well as the administrator. The user is responsible for applying a combination of the provided transformations, thereby creating new Internal Representations. The system is influenced by the actions done by the user and does not apply any changes of its own. Because the transformations cannot cause any changes in the File System, there is no need to grant the user special permissions.

## The life-cycle view of a system

ProFiler behaviour over time depends entirely on the actions of the user. The current system does not deal with changes that take place in the FS (This was defined as being out of the project's scope although it implies that the view displayed by ProFiler may not be valid).

**ProFiler life-cycle** describes the phases that the application goes through in a session with a user (Figure 4-5, 4-6 and 4-7):

1. **Initial setup:** The starting point for ProFiler is the extraction of the contents of the File System (via the Operating System).
2. **Creating the Internal Representation**: The default internal representation is **created** from the extracted FS data.
3. **Displaying the Internal Representation** as an External Representation: The internal representation is **displayed** to the user as an External Representation (ExRep) of the files and folders of the system. The combination of an Internal and External Representations create a **virtual view**. The view of the default internal representation represents the contents of the File System.
4. **Activating the transformations**: using Drag&Drop with the objects displayed in the external representation, the user can activate the above transformations
5. **Applying transformations**: The user applies a combination of the provided **transformations** to the default Internal Representation thereby creating new internal representations. This can be done successively.
6. **Storing the Internal Representation (InRep)**: The user can **save** any of the Internal Representations for later use.
7. **Retrieving a saved Internal Representation**: at any point the user may choose to restore a view he once saved. At this point ProFiler will restore the data and present this view to the user. The user can then apply further transformations to the InRep as in phase 4 above.

|  |
| --- |
| Figure ‎4‑5: The initial phases of the ProFiler application life-cycle where the FS contents are extracted as the starting point for creating virtual views of the file space |

|  |
| --- |
| Figure ‎4‑6: At any point in the creation of a virtual view it is possible to save a view for later use |

|  |
| --- |
| Figure ‎4‑7: The latter phases of the ProFiler application life-cycle – retrieval of a previously saved view and the subsequent transformations on it |

# SYSTEM DESIGN

This section discusses the design of ProFiler; it will explain the mapping of the architecture and its elements to the components that will implement and animate them.

An early design decision of the project was to implement the application logic as a local web service. This decision was the result of early discussions of aspects of the project that ultimately were not included in the architecture and implementation. We therefore decided not to include them in the requirement and specification sections.

## Data components

There are two major data structures that together make the ProFiler View (Figure 4-1) - the **Internal representation** and **External representation** or the **InRep** and **ExRep**.

### The Internal Representation (InRep) data structures

The **InRep** (Internal Representation) includes files, folders and the relationship between them in the form of a tree structure which keeps the relationships between them (5-1). The InRep uses three data structures: **Tree**, **File** and **Folder**.

The InRep List contains the current available views, stored as a list of InRep objects (Figure 5-2). The class diagram is shown in Figure 5.3.

|  |
| --- |
|  |

Figure ‎5‑1: The InRep List contains the internal representations of the created views. The views are made of Trees which are made of Folders which hold Files

### 

Figure ‎5‑2: The relationship among the Tree, File and Folder structures – components of the Internal Representation (InRep)

|  |
| --- |
|  |

### 

Figure ‎5‑3: Tree, File and Folder class diagrams – components of the Internal Representation (InRep)

|  |
| --- |
|  |

### The External Representation (ExRep) display

The **ExRep** (External Representation) is the representation of the InRep in a graphical form to the user (Figure 5-3). The InRep is a tree like structure that is transformed into a directory like structure by the library of **MIT [?]**.

|  |
| --- |
| Figure ‎5‑3: The display of ProFiler is made of the external representation of the currently available views |

## Process components

ProFiler is composed of several main components that carry out the main processes of the application: Transformations, Views, Extraction of files and folders, Save and Retrieve views and GUI display and activate processes. These are shown in a system state-transition diagrams (Figure 5-4):

1. **Extractor component**: is responsible for extracting the names of **files and folders** from the File System and to present them as the basic View of ProFiler in the web browser GUI.
2. **Creator** **component**: takes the default FS data and creates an InRep from it. When retrieving a saved View, it creates the appropriate InRep from it.
3. **Displayer** **component**: takes the InRep and transforms it to the ExRep – the representation as it appears in the graphical user interface (GUI). This together with a control panel and the Drag&Drop capability provide the user the ability to create, maintain and manage different view, and activates the above components.
4. **Activator** **component**: the control panel and the Drag&Drop capability displayed by the Displayer component provide the user the ability to create, maintain and manage different view, and activate the above components. These actions are translated into the relevant transformations.
5. **Transformer** **component**: is responsible for providing the user with the ability to create and modify a variety of points of view using **Filtering**, **Aggregating** and **Re-structuring** processes (see sub-section 3.1.3).
6. **Saver component:** takes the InRep data structure and serialises it so that it can be stored in the InRep Repository.
7. **Retriever component**: takes the serialised InRep representation chosen by the user from the repository and transforms it into an InRep data structure through the *Creator component*.

|  |
| --- |
|  |

Figure ‎5‑4: The overall design of ProFiler showing the components and the interactions among them - carrying out the processes, using the data and supporting the roles defined in the architecture

## Communication components (user-computer interaction)

## Interactions

This section will present different diagrams in order to describe the ProFiler design.

### State diagram

The ProFiler state diagram (Figure 5-5) represents the behaviour of an object during its life in response to user events, together with its responses and actions.

|  |
| --- |
|  |

Figure ‎5‑5: ProFiler state diagram

### Design sequence diagram

Figure 5-7 shows the possible branches of interaction between the user and the system.

|  |
| --- |
|  |

Figure ‎5‑6: ProFiler sequence diagram

## System Architecture

The second architecture you had in your SDD, with modules, separation of concerns, layers, services, etc. Updated to your implementation.

The overall design (shown in Figure 5.5) describes the components that are necessary to carry out the ProFiler processes and the data needed for those processes in order to support the user role. The configuration of the ProFiler system is the result of mapping the ProFiler design onto specific components using the 'Three tier software architecture'. The resulting configuration, shown in Figure 6-1, is composed of three tiers or layers:

* **Data layer** includes:
  + The resident **operating system** which provides access to the File System through a library that enables communication with the File System.
  + A **MongoDB database** which can store the Internal Representations (InRep) that the user created using the transformations and wishes to make persistent.
* **Application logic/functionality layer**: is implemented by a **local web-server** based on Node.js [Node.js 15] and additional libraries (Gulp [Gulp 15] and Bower [Bower 15]). The layer supports the various transformations (aggregations, filtering and re-structuring) in order to assemble different points of view. The saving of Internal Representations in the data layer and the extraction of the file/folder data from the File System.
* **Client presentation layer implemented in the web browser**: the graphical user interface (GUI) uses the **web-browser** as the input/output medium. This provides the display to the user and the mechanism for registering user actions as input.

|  |
| --- |
|  |

Figure 5-7: The ProFiler system configuration - the result of mapping the ProFiler design (as shown in Figure 5.5) onto the 'Three tier software architecture' and the related components

# IMPLEMENTATION

A **web application** or **web app** is a client-server software application in which the client (or user interface) runs in a web browser and the server is usually run on a remote server (but can also run locally). The possibility to develop a web application composed of the client side only is feasible, as it happens when developing applications for Chrome OS, Windows and others (some will argue that in these cases it is no longer a web application). One of the main advantages of browser-based applications is that they will run on any computer which has a fully-functional web browser.

The decision to implement the system as a web application was an early design decision. It was made primarily for the sake of acquiring experience with web application technology, for making ProFiler easily portable and for the strong support it has for developing advanced GUIs. In retrospect, it may have been better to choose a language with good support for GUI design such as Java, C#, python, etc. and build ProFiler as a stand-alone application.

The decision to implement ProFiler as a web application posed a problem: the application must have access to the File System which resides on the same computer as the Browser. However, getting the Browser to access local resources through the operating-system is not straightforward when the server is remote. This is basically a security restriction aimed at protecting the computer from server side software that can be downloaded onto the client. In addition, the Browser does not have easy access to an external database. Accessing the file system is possible using HTML5 File API and accessing remote databases accessible through HTTP requests is possible using AJAX. Nevertheless, there are still limitations File API was not fully implemented by all Browsers, some Browsers’ versions do not support it and we are therefore not free to perform all files related operations as when developing native applications and when accessing remote databases we are limited for very specific ones.

A possible solution that keeps the advantage of a web application but circumvents the security restriction is to run the server locally. The server software can then access any operating system functionality necessary and also manage the required database. The resulting hardware configuration on which the application runs is the user computer with a Browser, a local Server and a Database (Figure 6.1).

There are disadvantages to such a scheme: the application requires the setting up of a local server and ports conflicts might arise.

## Algorithm Design

Describe relevant algorithms implemented in the project, if they were not described in earlier sections.

## Interfaces

This section will present the user interface, which is the web-based application ProFiler. The section will include screenshots of the different screens of the GUI.

**First screen – default view**: the first screen presented is the default view (left side of the screen) of the files and folders as they are stored in the File System. The user can open or close levels of folders. This view will be present at all time for the user to use (Figure 6-3).

|  |
| --- |
|  |

Figure ‎6‑4: Default view (left side)

**Guide**: according to the user’s cookies, if this is the first time the user uses ProFiler a guide will be presented to him, explaining the main features and sections of the system. The guide will be always available using the ‘?’ button (Figure 6-4).

|  |
| --- |
| C:\Users\tcohen\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\Screen Shot 2015-08-04 at 10.45.17 PM.PNG |

Figure ‎6‑5: Guide button

**ProFiler sections:** profiler is divided to 3 sections: intermediate, working stations which the user can create, re-structure and search and the default view (left side) of the files and folders as they are stored in the File System (Figure 6-5).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑6: ProFiler sections

**ProFiler features:** profiler provides the user to drag & drop files, folders and trees, save and restore trees which created by the user (Figures 6-7 and Figure 6-8).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑7: Drag & Drop feature

|  |  |
| --- | --- |
| Screen Shot 2015-08-04 at 10 | Screen Shot 2015-08-04 at 10 |

Figure ‎6‑8: ‘Save tree’ button and ‘open and choose tree’ button

**Intermediate station**: in this section of the screen, the user can search\filter files and folders by typing the relevant keywords (Figure 6-9).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑9: Search/filter files and folders

**Working station**: the user can drag and drop files and folders (along with their content) to this section of the screen to build trees (Figure 6-10).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑10: Working station

**Saving a tree**: the user can save a tree for future work by using the save button. While saving the tree the user can name the tree and add comments (Figure 6-11).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑11: Saving a tree

**Restore saved tree**: the user can restore a tree he previously saved and continue modifying it. When the user opens a saved tree additional information is displayed on the right side of the screen (Figure 6-12 and Figure 6-13).

|  |
| --- |
| Screen Shot 2015-08-04 at 10 |

Figure ‎6‑12: Restore saved tree

|  |
| --- |
| C:\Users\tcohen\Downloads\Screen Shot 2015-08-04 at 10.53.45 PM.png |

Figure ‎6‑13: Restored tree

## Development environment

ProFiler project was developed on **PhpStorm** environment [PhpStorm 15].PhpStorm is a cross-platform IDE, providing an editor for several programming languages such as PHP, HTML and JavaScript with on-the-fly code analysis, error prevention and automated refactoring for PHP and JavaScript code, supporting SQL and database connection and the ability to add ships with pre-installed JavaScript plugins.

## Programming languages

Based on the selected technologies specified in the technology survey section, there are two parts of the project and each has its own environment, languages and libraries (Figure 6-3):

* The client-side (GUI) - this part is composed of components which were developed using:
  + HTML
  + CSS
  + JavaScript
  + AngularJS
* The backend side - this part is composed of components which were developed using:
  + Node.js

|  |
| --- |
| ‎6‑3: ProFiler layered system configuration with the libraries they use |

## Risk Management

This important section should include main risks in the projects, including priorities of the risks and the entire plan you used to manage these risks.

## Exceptions Management

This section includes erroring and exceptions handling.

## Versions Control

You may include here print screens of your versions control processes, or links to open version controls you used over the year.

## Project Management

This section includes tasks and time management. הגאנט שלכם, חלוקת עבודה ביניכם וכל ניהול הפרויקט כולל חלוקה לאלפא ולסוף פרויקט

## Code

This section includes links to GitHub or any other code repository. If you deliver the code in any other way, this section should include directions regarding where the code is and how to test it.

You may show examples of main code sections

1. Code to GitHub
2. Code to your recording of using the system
3. Code to your working system online
4. Readme if needed

# SYSTEM VALIDATION

פה עליכם להציג את דוח הוולידציה, כפי שהוצגה בפני השופטים, כולל פסקת הסבר, מה שיניתם, מסקנות ותיאור התהליך. במידה והמשכתם עם עוד וולידציה בקיץ, ניתן להוסיפה.

# SUMMARY, EVALUATION, CONCLUSIONS AND FUTURE WORK

## Summary

Working with files, folders or any other information source has a real purpose in our world; any information can serve different purposes at different times for different people. ProFiler is a PIM tool that provides the means to create, manage, organize and maintain the points of view the user needs. A user may want to look at files according to one classification and then look at the same set or a subset of those files using another classification.

ProFiler was developed in order to address this issue - to provide the user with the ability to create and modify different points of views of the Files/Directories File System. ProFiler’s architecture is designed to apply unlimited transformation on any set of hierarchical files and folders and to store the results for later use. In short - ProFiler is a first step in developing a personal information management tool that is focused on the representation of the different points of view.

The second generation of ProFiler should include several extensions to further assist people who use this application (see sub-section 7.2 - Future work). The basic issues relate to the fact that ProFiler creates alternative views of the File System. Those views and the original data in the File System can get out of step and become inconsistent.

## Evaluation and conclusions

No extensive field studies were conducted with ProFiler and this hinders a thorough and meaningful comparison of traditional browsers with ProFiler. However, from an individual point of view, ProFiler works well - allowing the user to construct multiple virtual views based on the initial default view of the File System (FS). Virtual views can be constructed by modifying the structure of the folders, by adding new folders with links among them and by populating them with some or all of the existing files from the FS. Using the Browser based GUI allows the user to exploit the 'drag&drop' mode of interaction in an intuitive manner.

ProFiler also allows saving a virtual view and retrieving it at a later point in time making the desired views persistent. In that respect ProFiler answers all the requirements agreed on in the specification phase of the project (Section 4.1).

When using the system, however, it becomes patently clear that an aspect that was left out of the project due to time and man-power constraints makes the system's usefulness limited and perhaps even hazardous. We refer to the following point from Section 4.2:

* **FS changes & consistency checks**: any changes in the FS caused by the user (through the use of the default FS browsers) which affect the views derived by ProFiler should be conveyed to the user. The ProFiler **will not be notified** of changes in the FS and the changes made in ProFiler will have no effect on the FS.

In retrospect, it appears to be an essential property of such a system since the changes to the FS may make the virtual views inconsistent and thereby misleading.

The decision to design and implement the system as a web application had some advantages and disadvantages. The advantage was the ease of GUI development providing a flexible system with an impressive interface. The main disadvantages are that there is a need for a local server and it requires an installation process. If such a system was to be built again in the future – we suggest looking carefully at the available technologies and considering building it as a stand-alone application based on a language such as Java, C#, Python, PHP, etc. without the use of a browser and a server.

## Future work

There are several issues which the current version of ProFiler does not deal with and that should be addressed if ProFiler is further developed:

* **Adding attributes**: Adding attributes to a file and adding the ability of sorting/structuring the alternative views according to those attributes. These attributes include: name, created date, comments, link to other files and classification.
* **Saving transformations**: saving of Transformation to be able to apply them (semi) automatically at a later stage. When there is a change in the File System the user can still apply the saved transformation on the update structure (Figure 7-1).
* **Open more types of files**: currently, the application can open WORD documents when they are double-clicked. In order to use ProFiler as a useful file browser that can be used more efficiently, the user should have the ability to open more types of files from the application (such as Doc, PPT, PNG etc.).
* **Modify the GUI**: in order to make ProFiler more popular, the GUI should be modified to have a Windows Explorer look and feel.
* **Consistency checks**: check for inconsistencies that arise from changes made to the File System by the default browser. Should inconsistencies be reflected in the ProFiler data/view and if so – how? Should the changes be made automatically or should the user be notified of any change – for example, when the user is restoring saved views (Figure 7-2.
* **ProFiler’s modifications:** allow updates in ProFiler data to be reflected in the File System (Figure 7-3). The consequences of such ability are not clear and therefore serious consideration is required before attempting to add it.

|  |
| --- |
| File System  ProFiler View  Transformation  Save  Apply |

Figure ‎7‑1: Saving transformation and being able to apply them later semi-automatically

|  |
| --- |
| Updates  File system  ProFiler View  Modifications |

Figure ‎7‑2: ProFiler views should be made aware of changes made to the File System - keeping things consistent may prove tricky

|  |
| --- |
| Updates  File system  ProFiler View  Modifications |

Figure ‎7‑3: Allowing specified modifications performed in ProFiler to affect the File System – an open question

# REFERENCES

**[Barreau 95]** Barreau, D. and Nardi, B. “*Finding and Reminding: File Organization from the Desktop*”, SIGCHI Bulletin, 27(3), July 1995.

[**Bower 15**] "*Install Bower packages*", 2015, <https://www.npmjs.com/package/gulp-bower>

**[Dourish 99]** Dourish, P. W., Edwards, K., LaMarca K. and Salisbury, M. "*Presto - An Experimental Architecture for Fluid Interactive Document Spaces*", Computer Science Laboratory, 1999.

**[C# 15]** "How to: Get Information About Files, Folders, and Drives (C# Programming Guide)" , 2015, [*https://msdn.microsoft.com/en-GB/library/6yk7a1b0.aspx*](https://msdn.microsoft.com/en-GB/library/6yk7a1b0.aspx)

**[Explorer 15]** *"Windows Explorer or File Explorer*", 2015, [*https://en.wikipedia.org/wiki/File\_Explorer*](https://en.wikipedia.org/wiki/File_Explorer)

[**Express** 15] *“Fast, un opinionated, minimalist web framework for Node.js”*, 2015, <http://expressjs.com/>

[**File Manager 15**] "*File manager*", 2015, <https://en.wikipedia.org/wiki/File_manager>

**[Gulp 15]** *“Automate and enhance your workflow for node.js”,* Wikipedia, 2015, <http://gulpjs.com/> and <https://www.npmjs.com/package/gulp-bower>

**[Node.js 14]** *"Node.js v0.12.7 Manual & Documentation"*,2014, <https://nodejs.org/api/>

**[Multi-tier 15]** *"Multi-Tier Architecture"*, 2015, <https://en.wikipedia.org/wiki/Multitier_architecture>

**[Mac Finder 15]** "*Switch Basics: Moving from Windows Explorer to the Finder*", Mac Finder, 2015, <https://support.apple.com/en-gb/HT201748>

**[Mac Finder 14]** "*Mac Finder*",2014,<https://en.wikipedia.org/wiki/Finder_(software)>

**[PhpStorm 15]**  *"Lightning-smart PHP IDE*", 2015, <https://www.jetbrains.com/phpstorm/>

**[Nautilus 15]**  *"Nautilus - Default File Manager in Ubuntu"*, 2015, <https://help.ubuntu.com/community/DefaultFileManager>

[**PIM 15**] "*Personal Information Management*", Wikipedia, 2015, <https://en.wikipedia.org/wiki/Personal_information_management>

**[Q-Dir 15]**  *Q-Dir website*, 2015, <http://www.softwareok.com/?seite=Freeware/Q-Dir>

**[Voit 09]** Voit, K., Andrews, K. & Slany, W., "*Why Personal Information Management (PIM) Technologies Are Not Widespread and What to do About It*", PIM 2009

# APPENDIX A: Background material

The **File Explorer** [Explorer 15] was introduced with Windows 95 (as Windows Explorer). It is a graphical file management utility for the Windows operating system. The goal of the file management utility is to allow users to conveniently work with the files and directories located on any connected disk. When a file is selected within the Windows/File Explorer interface, users can select from a variety of context-sensitive actions such as renaming, moving, copying, publishing, emailing, printing or deleting the file. Entire folders can also be opened, explored, shared, copied, cut, pasted, moved or deleted. Files can be searched or dragged and dropped between folders, and folders can be moved into other folders and disk drives as desired.

The **Mac Finder** [Mac Finder 14 & Mac Finder 15] is the default file manager and graphical user interface shell used on all Macintosh operating systems. The Finder uses a view of the file system that is rendered using a desktop metaphor; that is, the files and folders are represented as appropriate icons.

**Nautilus** is the default file manager in Ubuntu [Nautilus 15].

תקציר

סיירי הקבצים המסורתיים מספקים נקודת מבט אחת של מערכת הקבצים כפי שמשתקף בקבצים ובתיקיות שלהם. כל השינויים שיעשו לסידור של הקבצים והתיקיות כגון: מתן שמות או מחיקה של הקבצים והתיקיות יגרמו לשינויים במערכת הקבצים. גישה זו אינה לוקחת בחשבון את העובדה שאנשים ממלאים תפקידים שונים כשהם משתמשים במחשב ולכן עשויים לרצות להציג את האינפורמציה שלהם בדרכים שונות מבלי לארגן בהכרח מחדש את מערכת הקבצים שלהם בכל פעם שהם רוצים לעשות זאת.

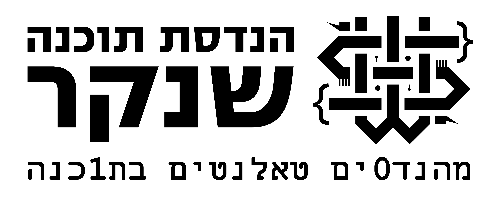
מערכת ProFiler היא סייר קבצים שפותח על מנת לספק למשתמש את היכולת ליצור ולשנות נקודות מבט על מערכתהקבצים מבלי לגרום לשינויים קבועים במערכת הקבצים. המשתמש מתחיל בחילוץ התוכן של מערכת הקבצים, ולאחר מכן על ידי יישום סדרה של טרנספורמציות כגון: סינון, צבירה וארגון מחדש של הקבצים והתיקיות, ניתן ליצור תצוגות שונות של נתונים המתאימים למטרות שונות.

הכרת תודה

ברצוני להודות לד"ר יגאל הופנר על ההדרכה, זמינות, העזרה, התמיכה והייעוץ שלו לכל האורך של הפרויקט.

לבסוף, ברצוני להודות לראש המחלקה ד"ר אמנון דקל ולצוות ההוראה והסגל המנהלי של המחלקה להנדסת תוכנה על תמיכתם במשך 4 השנים האחרונות.

|  |  |
| --- | --- |
|  |  |



|  |
| --- |
| **System ProFiler** |
| **הנדסת תוכנה B.Sc. ספר פרויקט גמר לתואר ראשון בוגר במדעים** |
|  |
| מאת: אילן אוקסנברג  מנחה:דר' יגאל הופנר |
| **‏07/01/2016** |

מוגש כחלק מהדרישות לקבלת תואר ראשון בוגר במדעים(B.Sc)