SCFE: Book of Specifications

Matthieu Stombellini
Mathieu Rivier
François Soulier
Rakhmatullo Rashidov

Contents

1 Introduction

Salamanders' Lab is a team composed of four motivated students who want to create a piece of software that would have for prime objective to be useful – we want to be able to use it in our daily life. One of the frustrating sides of using a terminal for people who are used to using a file explorer is file navigation. Indeed, constantly switching back and forth between "cd" and "ls" commands can be quite disorienting. Other tools are just not easily accessible without a file explorer, such as file selection and even a simple copy-paste of multiple different file easily.

This is why we decided to work on this project: a file explorer. "Salamanders' Console File Explorer", or SCFE for short, is a project which will attempt to reconcile power-users with a flexible and productivity-oriented interface that has lots of built-in capacities, and help new console users who wish to have a tool that resembles what they have already used before.

This project will not only be a file explorer, but we want to also integrate many features that will cater towards advanced users: git integration and shell commands compatibility (through a shell pass-through) are planned to be included in such a way that they will not hinder clarity and understandability. In order to get as much of a reach as possible, the file explorer will be console-based, and cross-platform compatibility is a requirement for us.

The project is an answer to a problem we all encounter when using our computers with a terminal. With that in mind, the ultimate goals of this project are to create a tool...

- that increases productivity and is simply useful
- that helps both power-users and newcomers
- that, by using a console interface and cross-platform abilities, is available on a vast majority of computers.
- that is easy to understand with adequate documentation, both inside the tool and with outside resources
- that will live on even after S2, if possible, thanks to its usefulness
- that just works, as it would be part of a daily workflow: it must break as little as possible. Bugs negate the ease of use and productivity aspects.

We understand that the tool may look easy to make, but the complexity will come from quality insurance and a very low tolerance for bugs or glitches, while creating a tool that both people who are not familiar with computers and absolute masters in IT can use efficiently.

2 The Team

2.1 Presentation of the team

Our team is composed of four people who want to help other's daily lives and who want to build something which is primarily going to be useful. Keeping the antecedent in mind, it is not hard to imagine that the idea of a game for this project was not exactly a match with the team.

With a great individual thrill for helping others, the team found its members really quickly. We wanted a group that would allow us to not only produce something great, but would also benefit from a great atmosphere of collaboration and communication. None of us found the idea of a game such a good way for creating something meaningful nor helpful for others. We therefore directed ourselves from the beginning towards the creation of a utility software that would complement our shared beliefs. Finally, we concretized the creation of our team. We wanted something that would help others as well as ourselves and we think this project is the perfect one to fulfil our great goal.

The name of the team, "Salamanders' Lab" obviously refers to the salamander, a black and yellow lizard-like animal. The reason for such an animal is its unusual colors that remind, in a way, a computer's terminal, with a black background and bright foreground colors, and also gives our logos a distinctive look and a recurring theme in all visual materials.

2.2 Presentation of the members

2.2.1 Matthieu Stombellini

I am Matthieu Stombellini, a student who loves creating tools of all kinds, with the hope of making them useful. I have quite a bit of experience in coding, although I am by no means an expert in the field – after all, I only have a few years of experience with Java, not much else. As the most seasoned member of the team, it only felt natural for me to be its leader. I am not one for absolute dictatorship, but I feel like I can be an adequate leader, with an important drive for constant improvement and quality. In short, I want to make something good that I will be proud of! I expect this project to give me a bit of experience in management capacities.

2.2.2 Mathieu Rivier

Having been raised in a family of computer scientists, with my dad and my step brother in the field, I was naturally introduced to some of computer science's essential concepts very early on. Later on, my attention turned towards the

managerial side of computer science, although I understand that, in order to be efficient at managing teams of people, I first need to learn and master fundamental ideas of computer science. In my eyes, this project represents a unique opportunity to improve the daily lives of the people around me, by helping them in improving their constant interactions with their files through the terminal, with the help of our team. Furthermore, I think that our group is the perfect one to conduct and realize this ambitious project, and truly believe that this experience will be extremely formative for the four of us both in terms of team work and programming skills.

2.2.3 François Soulier

My name is François Soulier, I'm 18 years old, and I am a student at EPITA. I do not have a lot of experience in coding, but it is an area I became really interested in the past couple of years, and I do have a real desire to learn about coding in every possible way. Nevertheless, like my comrades, the implementation of a video game did not really attract me very much, and the main reason for that is the risk to create a sort of copy of something that had already been made before. Also, we wanted to create a software that could be of a daily use, on which the use of Git could be easier and more efficient. This idea reinforced my motivation to get involved in the creation of a file manager. Finally, I find group projects very rewarding to work on a group project because it develops the ability to communicate with coworkers, which certainly is a skill to possess in the perspective of a future engineering life.

2.2.4 Rakhmatullo Rashidov

My name is Rakhmatullo. Last semester, I was an exchange student at EPITA and it was glorious experience in my life. For the past two and half years, I have been studying Computer Science and Engineering in South Korea and have learnt programming languages such as C, C++, Java, HTML and CSS. I love coding – I find being able to find a simple solution, with less time complexity, to a difficult problem to be particularly exciting. Since what I look for in life is knowledge, I wanted to stay at EPITA. My goal is to be a professional computer engineer and I believe that what I learn at EPITA will help me a lot. I had never worked on any project before, so I am truly excited about our project. I also believe that our project will be helpful for many people.

3 The Project

This project is different from the ones we heard were traditional to the second semester at EPITA. Indeed, none of us four really found the idea of a game

interesting or stimulating enough for such a project. It was nearly instantly that the team got together and only 3 days later, the idea was there: we would create a Console File Explorer.

At first glance, a file explorer might seem like an uninteresting choice of project. For us, creating meant building something that would complement the workflow of not only ourselves, but also EPITA students and more broadly any person enjoying the terminal who still wants more from it.

We all had a hard time dealing with the stock, inefficient file managers that come with our terminals, and we thought we could all considerably reduce the amount of time spent looking for files and greatly improve the happiness factor while having to deal with the file manager.

Therefore, we settled down for this idea of creating something meaningful in our sense and were thrilled to see that some of our classmates already showed interest in our project in a user perspective. We are furthermore excited to have already found such a great amount of feature implementation that could bring prospective users the best possible UI experience. This project will therefore be a **Utility Software** that we think will bring a little something to the world. We of course do not have the pretension of changing the world, but just helping the people around us with our own means.

3.1 State of the art

First of all, file managers have been around for a long time now. The appearance of the first ones goes back to the 80s. There were in fact directory editors in Windows. Later, with the release of Windows 95, came a 16-bit version of File Manager, which did not support extended file names. The software being rewritten again as a 32-bit version, the issue was fixed with Windows NT 3.1 up to 4.0. Also with Windows 95 came Windows File Explorer, which was included and provided a graphical interface for the user.

Nowadays, many file managers are used depending on the operating system. For instance, as graphic interfaces, there are Finder on MacOS and Windows File Explorer on Windows as said previously. As for Linux, many graphical GUIs are known, but the most broadly available technique is to use the terminal with `cd` and `ls` commands.

As a matter of fact, it is on Linux and UNIX operating systems that most of Console File Explorers can be found. As examples there exist Midnight Commander, a powerful orthodox file manager, and Ranger, which was written in Python. Nonetheless, no cross platform console file manager has been created yet. In point of fact, each of them that have been created until now are only mono- platform. It is rather important to underline the fact that a console is a really powerful platform, and that console file explorers are created to make the use of the console much easier for any kind of user.

This lack of a cross-platform manager is the precise reason why this project aims to create a new tool that can be used on any operating system.

3.2 Goals

A file explorer would be nothing without its numerous features. Here is a breakdown of what we want to focus on.

- Ease of use: the project has to be accessible to people who do not have much experience with terminals
- Beauty: one of the main challenges in this project is creating something that simply looks good – and doing this in the terminal will be rather difficult, but also very interesting.
- Usefulness: The project has to be of interest for all users and add something pleasant to our everyday workflow.
- Cross-platform: The project has to be cross-platform, both because two of the members of this group are Mac users and because we want to be able to use this tool more or less everywhere, for it to be as useful as possible.

In order to not fall into a "too hard to use and not useful enough" category, an additional goal is to be able to satisfy both the needs of power users (who just use "cd" commands all the time) as well as simple users who are more used to graphical utilities.

3.3 Features

A file explorer would be nothing without a solid set of features. Here is a breakdown of what we want to do.

3.3.1 Interface

The main idea of a file explorer being exploring files, the interface has to be focused on the files, and exclusively on them.

```
dir: ~/myproject/
--------------------------------------------------
  name              size      date        git
  ..                <parent dir>
  .gitignore        1 kB      2019-01-13  up-to-date
> hello.jpg         1 MB      3 min       up-to-date
  code.js           133 kB    13:45       changed
```

```
  data.xml          430 kB    1d          (ignored)
  data.json         250 kB    2019-01-15  unstaged
--------------------------------------------------
NAV | Ctrl+E to change mode
~> Ctrl+Enter to start typing a command
```

The mockup above is clear, concise and easy to understand, even for beginners. You can clearly see what the current directory is, what the files inside it are, which file is currently selected (identified by the `>` character), as well as the navigation mode (`NAV`, detailed in a later section) and the command and search buffer (after the `~>`). All of the functionality is available at the user's fingertips.

The interface is divided in a few sections:

1. The first one is the directory path, which is an absolute or relative path to the directory that is displayed in the file list
2. The second one is the file list (or file area), which displays either the list of files in the current directory, or a relevant submenu. The files themselves could eventually use color codes for more information (much like `ls` displays different colors for folders).
3. The last one is the status area, which has the name of the current navigation mode, a message zone, and the input area. The message zone may contain any relevant information, which includes important shortcuts by default. The first message it should display the "Ctrl+E to change mode" text, as it is the most important command when a user starts the software. Each mode uses the input area differently, whenever they may need the user's input.

```
MODE | Message area
~> Input area
```

The files can be sorted in different ways (by name, by modification date...) through the Ctrl+T shortcut to change the sorting method and Ctrl+O to change the order (increasing or decreasing). The following sorting methods should be available:

- By name, handling numbers in a "smart" way, where "text3.txt" is considered to be before "text10.txt" when most file explorers wrongly puts the former after the latter.
- By size, using the file size computed by the underlying system.

- By type, using file extensions, considering that folders come before files
- By date, using the last modification date

When changing the sorting method or order, a reminder of the method and order used will be displayed in the Message Area.

The columns displaying the files' properties (modification date, git status, size...) should be modifiable in a config file, with both order and display importance. Some columns should have an additional "compact" mode that only takes up a minimal of size, without sacrificing clarity.

3.3.2 File navigation

There will be a few ways to navigate through the files, using "modes". A mode is a way for the program to handle user input, changing how it reacts to input from the user. One can change between the NAV and SEA modes with the Ctrl+E shortcut, while accessing the COM mode is done with the Ctrl+Enter shortcut.

3.3.2.1 NAVigation Mode

The `NAV` mode provides basic navigation with arrow keys and letters. Letters follow the same order for vim line navigation: `H` to go to the left (or cancel), `J` to go downwards, `K` to go upwards, `L` to go to the right (or confirm). This allows for quick and easy navigation when the fingers are on the home row. Using the Shift key with J, K or an arrow key makes the navigation faster, by going down 10 files instead of 1.

Shortcuts like Ctrl+C to copy, Ctrl+X to cut, Ctrl+V to paste can be used without using the Ctrl key: pressing `C` will copy the file, `X` will cut it, etc.

The NAV mode is the default mode used when the program is started. It does not use the Input Area, which only displays a message telling the user how to run a command.

3.3.2.2 SEArch Mode

The `SEA` mode is another way to navigate through files: through simple file searching. The user can type the beginning of a file's name to automatically select it. In the interface given above, starting to type "cod" would automatically select the "code.js" file. Arrow keys can be used to navigate up and down. The Input Area displays the current search terms.

3.3.2.3 COMmand mode

The `COM` mode is a special mode that can be entered by pressing Ctrl+Enter. The text box at the bottom of the terminal will then be focused, and commands can be entered. These commands are passed onto a shell. This allows power users to enter custom commands and perform actions that might not have been possible with the GUI.

```
COM | Esc to cancel, Enter to run
~> cp myfolder/* destination/
```

This can also be used to change the directory shown in the interface to any directory.

```
COM | Esc to cancel, Enter to run
~> cd /my/favorite/folder/is/here
```

Leaving the `COM` mode is done by simply pressing the Escape key.

3.3.3 File manipulation

A file can be opened by simply pressing Enter while it is focused. The focused file can be recognized by the arrow (`>`) on its left. In the example given in the Interface section, pressing Enter would open the `hello.jpg` file using the an application that depends on the system's settings. Pressing Enter on a directory opens this directory.

More complex file operations can be performed through a submenu, which can be opened using Shift+Enter:

```
* hello.jpg        1 MB      3 min       up-to-date
    | > Open [Enter]
    |   Delete [D]
    |   Copy [C]
    |   Cut [X]
    |   Select/Deselect [S]
    |   Add to git [G]
    |   --------------------
    |   Back to folder [Q] or [Esc]
```

The entries in this submenu are self-explanatory. The user can navigate through them using the arrow keys or H, J, K and L (even if the mode is `SEA`). The user can jump to an option using the letter displayed between brackets. These letters also correspond to the Ctrl+(letter) shortcut, as well as the

letter shortcuts in `NAV` mode. This provides an easy reminder for complicated shortcuts, negating the overcomplicated letter shortcuts that can be seen in applications like `emacs`. The submenu can be escape using the `Q` or Escape key.

The interface itself can either be a page that takes up the entire file area, or a popup menu, depending on what the clearest solution ends

Files can also be "selected": this is equivalent to using Ctrl+Click on a traditional graphical file explorer. Selection can be done either through the "Select/Deselect" menu option, or through the Ctrl+S (or just S in `NAV` mode) shortcut. Operations like copying, cutting and deleting will act on all selected files and folders. An alert prompt will appear on all destructive operations.

All operations that take more than a couple of seconds will open a full-screen blocking prompt in order to ensure the user is aware of a background operation. This might be changed in the future for a more robust background tasks manager if deemed necessary.

3.3.4 Git integration

Part of a developer's daily workflow involves being able to interact with git repositories and this experience can be somewhat frustrating and even quite repetitive, a typical workflow involving

```
1. git add myfile.txt
2. git commit "Modified my text file"
3. git push
```

Each of those 3 commands, have their own variations so it just stacks up to a consequent amount of commands to type every time making the process adaptive but maybe not so efficient.

Git integration takes the form of

- The addition of shortcuts specific to git actions
- The addition of a "git status" column in the interface

Git should be as integrated as possible, without going overboard with useless features. If the user wants to perform more complicated commands, they should use either the `COM` mode or a separate terminal or application.

3.3.5 Documentation

While documentation is already a requirement in the scope of the S2 Project, we plan on making it a standout feature of our program. Because we intend

to develop an application that is ran in the console, informing the user on what he can or cannot do is difficult when only in the GUI. We intend to focus on documenting features immediately and producing an easy-to-read documentation for our program.

3.3.6 Miscellaneous UX features

These features are fairly important but are not key components of the app – they are only really relevant to the user's experience.

3.3.6.1 Color codes

Color codes should be used in the interface with a few different scheme available.

- Simple scheme: color folders and files depending on their file extension
- Git scheme: color files depending on whether they are tracked, staged, up-to-date,etc. For example: yellow/orange for a file that has unstaged changes, green for a file that is staged...

While they are not the main concern, the core interface APIs need to take colors into account. Color should also help the user, not make them blind.

3.3.6.2 Shortcuts

Because of the lack of mouse support in a simple terminal, shortcuts are a necessity.

The list of shortcuts will be expanded and detailed as the project is developed, but many commands are essential, such as copy, cut, paste... Some shortcuts were already mentioned in earlier sections. The idea is that the user should be able to navigate through all of the interface without either touching the arrow keys nor the mouse.

3.4 Technical details

This section will mostly answer the "How" question, discussing the technical side of our project.

This project will be coded in C#, as it feels like the most appropriate language for this task. The interface itself will be coded either using the basic `Console` API or using a more complex library like gui.cs or CursesSharp. Handling file manipulations will be attempted with the core API as much as

possible, creating our own algorithms, but if some operations end up being too difficult, we might have to use third party APIs to facilitate file manipulation.

While the idea of the tool is simple, we want to make it as cleanly as we can, ensuring proper functioning at all times. This implies quite a time cost that will be spent on testing the tool. We do not expect any economic costs.

3.5 Tasks distribution

The following tables show how we intend to distribute the different tasks

Task	Matthieu S.	Mathieu R.	François S.	Rakhmatullo R.
Core UI*	In charge		Substitute	
Navigation modes	Substitute		In charge	
File I/O		Substitute	In charge	
User input		In charge		Substitute
Tool & OS integration	Substitute			In charge
Website		In charge		Substitute
Documentation	Substitute			In charge

(Core UI corresponds to low-level interface work. All of the other tasks will use the Core UI to create and manipulate interfaces)

The following table shows the intended planning. This planning is created based on which tasks depend on others.

Task	Presentation 1	Presentation 2	Final Presentation
Core UI	70%	90%	100%
Navigation modes	20%	70%	100%
File I/O	30%	70%	100%
User input	40%	50%	100%
Tool & OS integration	0%	30%	100%
Website	0%	50%	100%
Documentation	20%	50%	100%