Uspector: A web platform to support design workflow management and formal evaluation of prototypes

Georgios Topsis

Thesis submitted in partial fulfilment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Professor *Constantine Stephanidis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Uspector: A web platform to support design workflow management and formal evaluation of prototypes.

Thesis submitted by
Georgios Topsis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Georgios Topsis

Committee approvals: _____
Constantine Stephanidis
Professor, Thesis Supervisor

_____
Dimitris Plexousakis
Professor, Committee Member

_____
Dimitris Grammenos
Principal Researcher, Committee Member

Departmental approval: _____
Antonios Argyros
Professor, Director of Graduate Studies

Heraklion, November 2019

Uspector: A web platform to support design workflow management and formal evaluation of prototypes.

Abstract

Designers use their skill set to develop usable and aesthetically pleasing products that are effective at addressing users' needs. Among the available product design approaches, User-Centred Design (UCD) is the prevailing one, as it focuses on optimising the product around the users' abilities, requirements and preferences, rather than requiring them to change their behaviour to accommodate it. By design UCD is an iterative process, thus designers often struggle to deal with the continuous management and usability testing of the designed mockups. After each iteration, designers need not only to revise their mockups based on the findings and insights emerging from user testing, but also to centrally organise them so as to manage new series of usability tests that will assess the new designs. This is a resource-consuming process for design teams and project managers.

This thesis presents Uspector, a web platform that aims to accommodate the needs of designers, practitioners and evaluators during the iterative user-interface design process of a product. In particular, through Uspector: (i) designers can organise their creative content in projects of screens, with every screen consisting of a set of variations, and they can keep track of their evolution over time; (ii) practitioners can efficiently organise and conduct usability experiments using the Heuristic Evaluation methodology in order to uncover possible usability issues of designs; and (iii) evaluators can be supported while inspecting designs for errors.

Uspector has been tested in a case study of iterative design. The case study has confirmed that the platform could eventually assist designers to centralise and organise their creative content, uncover possible usability issues by conducting multiple low-cost and rapid usability tests, as well as inspect the evolution of their work through Uspector's version control system.

Uspector: Μια διαδικτυακή πλατφόρμα για την υποστήριξη της διαχείρισης της ροής σχεδιασμού και υποβοήθησης της διαδικασίας αξιολόγησης πρωτότυπων

Περίληψη

Οι σχεδιαστές χρησιμοποιούν τις ικανότητές τους για την ανάπτυξη εύχρηστων και αισθητικά ευχάριστων προϊόντων που ικανοποιούν τις ανάγκες των χρηστών. Ανάμεσα στις διαθέσιμες προσεγγίσεις σχεδιασμού προϊόντων, η Ανθρωποκεντρική Σχεδίαση (UCD) είναι η επικρατούσα καθότι εστιάζει στη βελτιστοποίηση του προϊόντος γύρω από τις ικανότητες, τις απαιτήσεις και τις επιθυμίες των χρηστών, αντί να τους αναγκάσει να αλλάξουν τη συμπεριφορά τους για να το αξιοποιήσουν. Εγγενώς, η Ανθρωποκεντρική Σχεδίαση είναι μια επαναληπτική διαδικασία, έτσι οι σχεδιαστές συχνά αντιμετωπίζουν δυσκολίες στη συνεχή διαχείριση και τον έλεγχο ευχρηστίας των παραγόμενων mockups. Μετά από κάθε επανάληψη, οι σχεδιαστές χρειάζεται όχι μόνο να αναθεωρήσουν τα mockups τους με βάση τα ευρήματα και τις ιδέες που προκύπτουν από τις δοκιμές ευχρηστίας, αλλά και να τα συγκεντρώσουν και οργανώσουν κατάλληλα ώστε να διαχειριστούν μια νέα σειρά δοκιμών ευχρηστίας για την αξιολόγηση τους.

Αυτή είναι συχνά μια απαιτητική και χρονοβόρα διαδικασία για τις ομάδες σχεδιασμού και τους διαχειριστές των projects.

Η παρούσα διπλωματική εργασία παρουσιάζει τη διαδικτυακή πλατφόρμα Uspector, η οποία στοχεύει στην κάλυψη των αναγκών των σχεδιαστών, των επαγγελματιών (practitioners) και των αξιολογητών δοκιμών ευχρηστίας κατά την επαναληπτική διαδικασία σχεδιασμού των διεπαφών ενός προϊόντος. Ιδιαίτερα, μέσω του Uspector: (i) Οι σχεδιαστές μπορούν να οργανώσουν το δημιουργικό τους περιεχόμενο σε projects από οθόνες, με κάθε οθόνη να αποτελείται από μια σειρά εναλλακτικών εκδόσεων, καθώς επίσης και να παρακολουθούν την εξέλιξή τους με την πάροδο του χρόνου. (ii) Οι practitioners μπορούν να οργανώσουν και να διεξάγουν αποτελεσματικά πειράματα ευχρηστίας χρησιμοποιώντας την μεθοδολογία της ευρετικής αξιολόγησης προκειμένου να αποκαλυφθούν πιθανά ζητήματα ευχρηστίας των σχεδίων τους. (iii) οι αξιολογητές μπορούν να υποστηριχθούν κατά την αξιολόγηση της ευχρηστίας των σχεδίων για σφάλματα.

Αφού πραγματοποιήσαμε μια αξιολόγηση χρηστικότητας στο Uspector, προέκυψε ότι, στο πλαίσιο της διαδικασίας της Επαναληπτικής Σχεδίασης η πλατφόρμα θα μπορούσε τελικά να βοηθήσει τους σχεδιαστές να συγκεντρώσουν και να οργανώσουν το δημιουργικό τους περιεχόμενο, να αποκαλύψουν πιθανά ζητήματα ευχρηστίας με τη διεξαγωγή πολλαπλών, χαμηλού κόστους και αποδοτικών, δοκιμών ευχρηστίας καθώς και να επιθεωρούν την εξέλιξη των σχεδίων τους μέσω του συστήματος version control του Uspector.

Ευχαριστίες

Ευχαριστώ την Κατερίνα, για την διάθεση της να βοηθήσει στην ανάπτυξη της παρούσας αναφοράς.
Επίσης, ευχαριστώ ιδιαίτερα το Ινστιτούτο Πληροφορικής του ΙΤΕ και το Ίδρυμα Ωνάση για την αναγνώριση και ενίσχυση της προσπάθειας μου κατά τη διάρκεια των σπουδών μου.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview - Setting the problem

Nowadays, one of the most critical success factors of software applications is the usability [37]. Given the current market in which there are several alternatives for a specific type of software, if the system is hard to use, the users will look for another application that let them to achieve their goals easily. Therefore, project managers and design teams should develop a product under effective, low-cost and flexible processes which allow them to constantly produce and validate new ideas. As a design team become bigger and more in-homogeneous, the need of a solid workflow management method and a fast validation process of new ideas is becoming vital for the development of a software product.

In the intermediate stages of User-Centred Design (UCD) process, designers typically produce a large number of design artefacts: site maps, story boards, static mockups, interactive prototypes, etc. [48]. Although a number of surveys and empirical evidence have shown that designers need better tools to manage and evaluate these design artefacts [48][40][43], designers are still using classic file storage and sharing services as well as ad-hoc versioning solutions, like manually renaming files [48][33]. This may be for two reasons: first, there is not a digital assets management (DAM) system, oriented fully on designer's workflow needs (management of mockups' revisions and variations) and second, there are no appropriate and effective methods for comparing, and find the differences/changes, between two static mockups (revisions of the same design), which are visualised as bitmap images. Although, a portion of the design tools have define their own file formats (i.e Sketch, psd), non of them is widely used by design teams due to their dependencies on specific Operating System (OS) or tools (i.e Sketch, Photoshop).

During each design iteration, and in particular after the creation of a design, the need for feedback from other designers, usability experts or end users is rises. This implies that a usability assessment of new designs should take place from a practitioner. That continuous need for validation of new ideas leads to a high cost on development's time and budget of a project/product. The major drivers of this

cost are the technology platform, the recruitment/honorariums of the participants, and the facilitation and analysis of test's results. Moreover, we need a usability evaluation methodology which can be applied not only during the design phase of a product, but also after a product has been implemented and launched in the market.

We created Uspector platform as a web-based solution to the aforementioned issues of design workflow management and continuous usability evaluation of designs. To manage design workflow, Uspector firstly ask from designers to upload their work as static images. Label new designs as screens, variations or versions according to the relation of each one with the other designs of the product. Subsequently, users of our platform can operate as practitioners by conducting usability tests following the Heuristic Evaluation (HE) methodology, in order to uncover possible usability issues of their designs. During the set-up stage of a test its practitioner is able to manage the deadlines of each evaluation phase (evaluation phases of HE), use a number of heuristics collections (Nielsen's and custom ones), include or exclude specific designs to test and of course, recruit other users to participate as evaluators on the test. During the evaluation session the evaluators have the ability to report (*Phase 1*) and subsequently rate the severity (*Phase 2*) of all reported usability issues. After the completion of a usability assessment its practitioner is able to analyse the results in order to export useful insights about the current designs of a product. Based on these insights, a designer is able to revise accordingly one or more designs that have been inspected on the test and re-upload them on Uspector. Finally, Uspector through its design version and comparison system track and records all designs' changes as well as maintains a backup for each version, and so it gives the ability to designers to inspect the evolution of their work.

1.2 Thesis Structure

The rest of this master thesis is organised in seven (7) main sections as indicated in the table of contents:

- Chapter 2 will introduce the current state of the art in relevant topics that this thesis is based upon. Firstly, it will review the literature regarding design version control systems and heuristic evaluation. Lastly, it will present a small review and comparison of existing systems with Uspector.
- Chapter 3 will report the functional requirements of each system's feature which were gathered for designing the system. Also, it will present the terminology (the vocabulary used in Uspector).
- Chapter 4 will dive into the high-level architecture (front and back-end) and the technical aspects (technologies) of the platform. Moreover, it will illustrate the conceptual model and the representation of the data that was adopted.

- Chapter 5 provides a tour in Uspector platform. It presents most the most important views by analysing each visual component of system's user interfaces. It answers the questions of how these views comply with the defining system's features in chapter 2 as well as how end-users interact with them.
- Chapter 6 will report the evaluation process, the scenarios utilised as well as the findings that were obtained.
- Finally, chapter 7 will summarise this work and discuss future directions.

Chapter 2

Literature review

Heuristic Evaluation

Heuristic Evaluation (HE) is the most popular of usability inspection method developed by Jacob Nielsen together with Rolf Molich 1990 [46] [42]. Indeed, it is well known across many fields: from Human Computer Interaction (HCI) and usability engineering to graphic design, visual design, information architecture, and to developers and project managers. The strength of the method is that it is quick, cheap, and easy to do. It is usually carried out by a single or a few evaluators who are equipped with just ten (10) usability heuristics [44]. This set of heuristics describe ten general principles for interaction design and they are used by the evaluators to examine a user interface and identify usability issues. However, though simple to execute, the method is open-ended and easily leads to unreliable results, i.e. there is little agreement in the problems reported by different evaluators of the same interface and high number of false alarms or problems that are missed [29]. Nevertheless, these weaknesses do not seem to reduce its popularity [31]. Increasingly, it appears that both usability specialists and non-specialists, who are not trained to carry out usability evaluations, use the method at some point during the development cycle. Finding ways to improve the method's performance is therefore practical and desirable.

To achieve better evaluation result with the HE methods on various information technology systems, many studies have been conducted to extend the traditional HE method in different ways [41]. Some differ in the heuristics they employ, some enhance the collaboration among participants [49], and others such as HE-Plus [34] [35] add a con-textualized layer called "usability problem profile" to aid the evaluation process. Our work use an extended version as well, given the fact that it allows practitioners to define their own collections of heuristics so that they have the flexibility to uncover usability issues for a specific product's domain (i.e apps for mobile devices, ambient environments etc).

In the the following subsection we present the comprehensive comparison study that was conducted to determine if the Uspector could produce results just as effective on existing works and how Uspector constitutes a more complete solution

to designers who need a more streamlined workflow.

Existing systems

Our research and analysis on existing works in the areas of design workflow management and usability inspection tools focused on working state-of-the-art systems from the domain of industry rather than of literature.

Our study based on two major system's characteristics, the support of a mechanism for designs' variations and versions as well as the conduction of usability inspection experiments using the HE methodology. In total, we concluded in a set of **nineteen** tools, fourteen (14) of which are Digital Assets Management (DAM) systems which support design workflow management (design variations and/or design version control systems) and the rest five (5) systems are usability testing applications which assist evaluators to perform usability inspections on their designs, using the Heuristic Evaluation (HE) methodology.

For the purpose of the comparison between Uspector platform and any of the existing systems, we extracted a set of attributes/requirements for each one of the two desired features (design workflow management, conduction of usability test using HE methodology) that a tool should support. Obviously, the categorisation of a software product/application can differ from the perspective it is studied. So, it is obvious that above systems support many more features which we did not take into account on our analysis.

Digital Assets Management platforms that support control systems for designs' Variations and Versions

A Digital Asset Management system is a solution for storing, managing and sharing digital assets among different members of a team. This kind of services can provide an appropriate environment for designers to store, manage, share and track the changes among mockups' revisions during the iterative design process (UCD methodology) of a software product.

In particular, our analysis focused on DAM systems which (i) are platform agnostic and (ii) support design workflow of a designer. Therefore, we concluded on the following set of criteria for systems comparison.

- Version History
- Version Comparison Mechanism
- Designs Variations support
- Platforms
- Files Supported

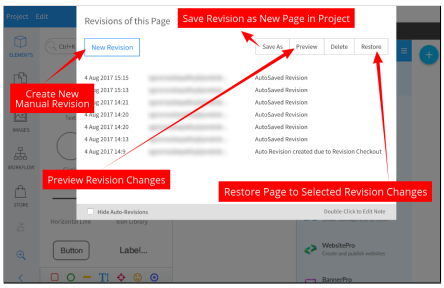
Below, we present briefly each tool as well as their differences with Uspector platform.

1. Git
   Git [7] is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It provides a version history for bitmap images or vector files (Sketch, svg etc), but its "diff" tool does not offer meaningful insights due to the fact that those file types contains vectors and not text.

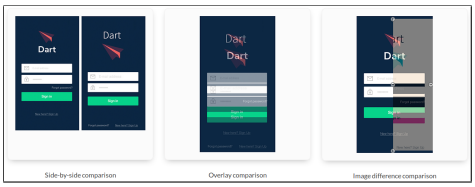
   Git is primarily designed for code (as in: text files) rather than designs (as in: sketch files and pictures) and it follows the software development workflow instead of the design one. Moreover, in comparison with Uspector platform it does not support the notion of design alternative.

2. Mockflow
   Mockflow [26] is an web-based solution for planning UI/UI, visualising user interfaces, creating user flows, documenting styles and approving designs.

   It provides a revision history (figure 2.1a) and a visual comparison system (figure 2.1b) with three modes to designers who want to highlight the changes between two different design revisions. Nevertheless, it does not support design alternatives as Uspector platform does. Moreover, Mockflow supports only designs with its native file format, a format which is generated by its editor for mockups creation, instead of being file format agnostic like Uspector (accept bitmap images).



(a) Revision history



(b) Diff tool between revisions

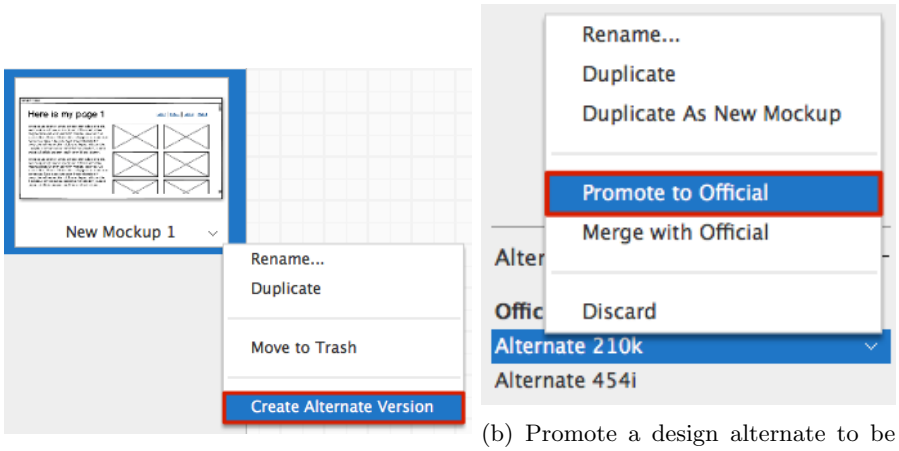
Figure 2.1: Mockflow tool

3. Balsamic
   Balsamic [22] is a wireframing tool which supports the creation and management of mockups variations (represent the idea of multiple variations of a same page/screen) but it lacks a VCS for designers

   As shown in figures 2.2a and 2.2b, Balsamic supports the creation and management of design alternates. Moreover, like Uspector, a designer can promote one of them as the *Official* one and keep revise it until it is approved

for development. A characteristic that differentiate from Uspector on this feature, is the support of the merging functionality between two design alternates. Balsamic keep only one design with the union of all changes.

Unlike Uspector, balsamic app does not support control systems for tracking (versioning) and comparing designers work.



(a) Creation of a design Alternate

(b) Promote a design alternate to be the Official one

Figure 2.2: Balsamic tool

4. Follio
   Folio [6] is a Mac only app, based on Git; with fair enough options as source files(sketch, photoshop, illustrator and SVG). Organising files is a bit messy as everything is at the same place and you cannot structure it. Nevertheless, it integrates with existing version control systems like github, bitbucket etc. Moreover, Follio allow teammates of a team to create and share versions (figure 2.3) of their designs and automatically keep them in sync.

   Unlike Uspector, Follio does not support the management of design alternatives and it is dependent on specific file formats (Sketch, psd, AI), a decision of the app which exclude designers who do not use Sketch, Photoshop or Adobe suite tools for the creation of their designs.

5. Invision app
   InVision app [9] is a prototyping tool created for designers, by designers. It allows designers to quickly and easily create interactive prototypes for your designs.

   This system allow designers to track changes on their designs and roll-back on any of them via an intuitive version history system. Moreover, as shown in figure 2.4 Invision app also offer a visual "diff" tool to designers to inspect the differences between two revisions of a UI mockup. Although Invision app is kind the norm in the preferences of designers, it still does not give

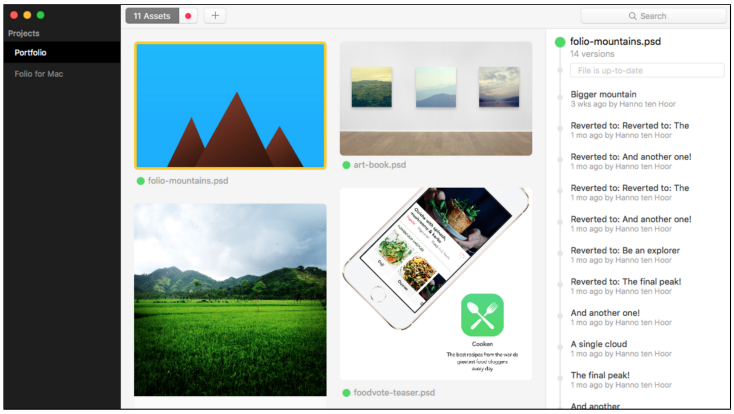


Figure 2.3: Design versioning in Follio desktop app

the ability to a designer to define two designs as alternative mockups, like Uspector platform does.

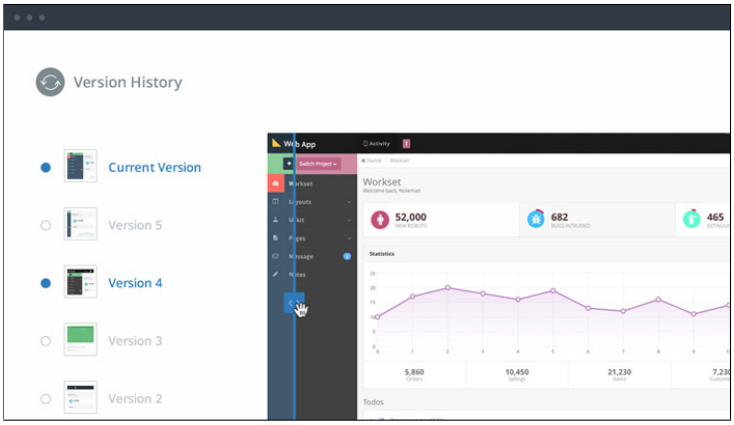


Figure 2.4: Design versioning in Invision

6. Figma
   Figma [5] is the a UI design tool that was expected to run through a web browser. It automatically saves versions of designer's work and allows its management at a later time (figure 2.5).

   Although, Figma is currently popular among designers, tool for creating and prototyping user experiences it does not allow a designer to create and manage variations of a design without adding more mockups to your project. That is a limitation which restrict designers from "connect" two designs, refine their ideas, and proceed, in the iterative design process, with one of them.

7. Kaktus

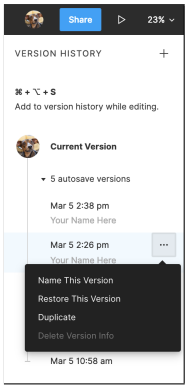


Figure 2.5: Design versioning in Figma

Kaktus [10] is a Mac-only desktop application that introduces a version control system and an effective visual "diff" tool, for Sketch files only. Designers using Kaktus are able to manage changes, document their work and keep their team in sync. It streamlines design workflow by offer to designers the experience of Git, with branches and commits.

Of course all of them are possible due to the nature of the comparable format [21] of Sketch files. In comparison with Uspector, Kaktus (i) is not platform agnostic, cause it is available only on Mac OS, (ii) it is not available to every designer, cause it supports only Sketch files and (iii) it does not support the functionality of design variations management, like Balsamic and Uspector tools do.

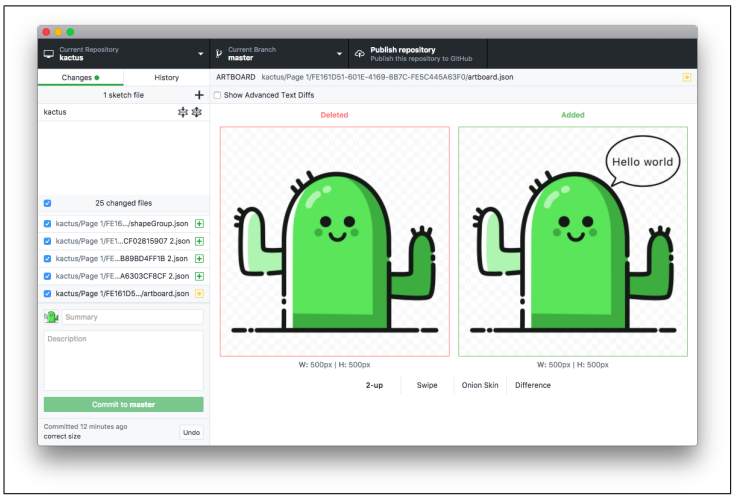


Figure 2.6: Design versioning in Kaktus

8. Plant app
   Plant [27] is a Mac app and Sketch plugin offering complete version control for designers and teams. In particular, Plant keeps a version history and allows the visually comparison between any two versions (figure 2.7).

   In comparison with Uspector, Plant is a Mac only app which does not integrate with other version control systems like git.It also supports only Sketch files and so designers should use the Sketch app in order to create their design assets. One more limitation of Plant app is that it does not allow designers to create design variations on a single mockup. Those core limitations differentiate Uspector and make it a more complete solution for designers.

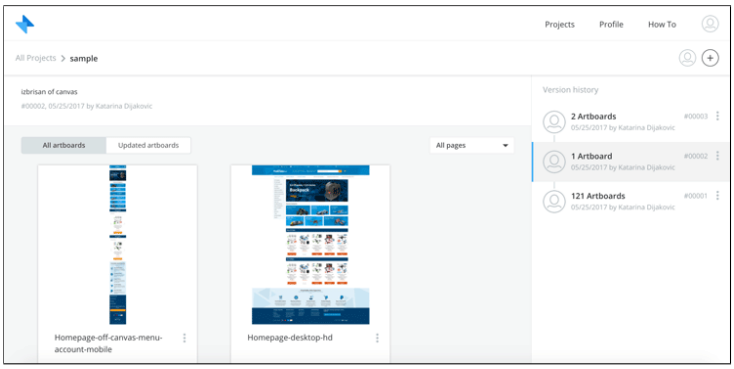
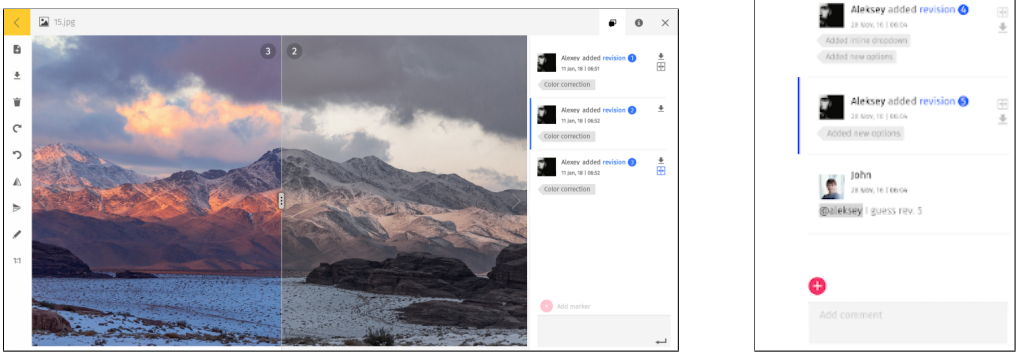


Figure 2.7: Design versioning in Plant app

9. Pics.io
   Pics.io [13] is a complete solution for designers to manage and distribute their digital content - on top of Google Drive. It offers a simple and straightforward version control system (figure 2.8b) to designers in order to track the changes of their designs after a revision cycle. Moreover, Pics.io offers a visual comparison tool for images which allow a designer to pick and compare, side by side, whichever two different versions of the same screen (figure 2.8a). Additionally, a designer can also approve or disapprove a specific revision does not fulfil the desired requirements/expectations.

   Although Pics.io provides a more intuitive and complete solution to designers to compare and find the differences between different revisions of a design () than Uspector, it still lacks the support of design variations which allow designers to validate their different ideas for a specific screen/page/mockup of a system.

10. Abstract
    Abstract [14] is a Mac only app which assists designers to manage, collaborate and version their Sketch files. Its version control for designers supports a modern design workflow. It tries to simulate the development workflow



(a) Visual comparison of revisions

(b) Design versioning with labels

Figure 2.8: Pics.io tool

that git offers with branches, commits and merges into the equivalent design workflow for designers.

Nevertheless, compared to Uspector, it is not platform agnostic and it supports only Sketch files, given the fact that it depends on Mac OS and Sketch. Also, currently it does not support the creation and management of designs variations.

11. Trunk
    Trunk [16] is a Mac only desktop app which offer an automated version control system for designers. It supports only Sketch, Photoshop and Illustration files which is a limitation to designers who do not include any of these files in their tool stack. Moreover, Trunk has been designed as a solution to design versioning without take care about the management of design variations by a designer.

12. Playbook
    Playbook [47] has been designed to be a revision control and comparison system for interactive mockups. It does not compare static images, but the interactivity of two designs. More specifically, Playbook assist designers to track the changes of two design revisions in terms of how the end users interact on them rather than what they actually see.

    Although Playbook assist a "diff" tool for design revisions, its approach is totally different than Uspector which aims to find the differences of two design revisions by comparing visually. Moreover, Uspector provides a mechanism to designers for the creation and management of design variations.

13. Versions by Sympli
    Versions [19], is a version control tool (figure 2.9) for designers with visual difference tool, merge and conflict resolution. Versions is based on Git but with a visual interface you can use to version and share your design work. Works with GitHub, Bitbucket, GitLab and Azure Devops. Versions compared to Uspector is not platform agnostic and it does support only Sketch file-type. Moreover, is does not allow designers to create and manage design variations.

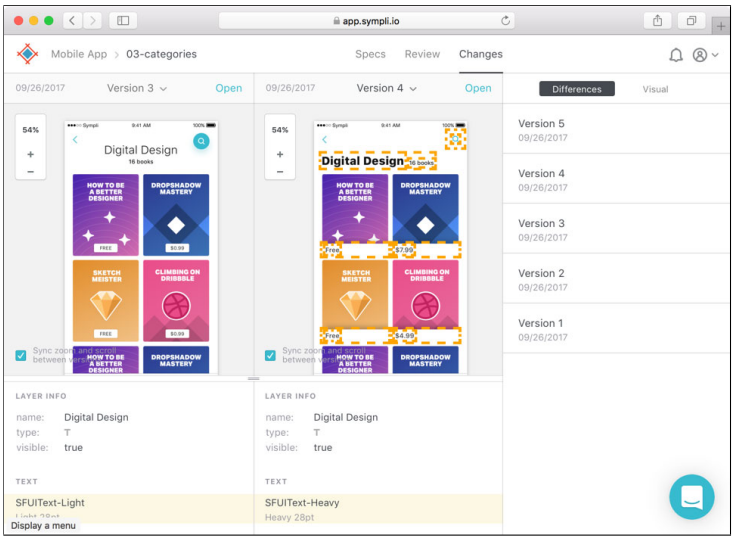


Figure 2.9: Design versioning in tool Versions by Sympli.io

14. GoVisually
    GoVisually [24] is a web app similar to Uspector. It also supports designers with a version control system, a feedback tool and it accepts designs as bitmap images. Nevertheless, designers cannot refine their ideas about a screen/page using design variations.

A summary of the comparison between Uspector platform and the above existing systems is shown in table 2.1. Each column of the table represents one requirement and each row a usability testing tool. Subsequently, we present a brief review of offerings of Uspector, against existing works, as a DAM platform which support design versions and variations.

Table 2.1: Comparison of design management systems which support design versioning

	Version History	**Version Comparison**	**Platform**	**Files supported**	**Design Variations support**
Git	Yes	No(CM)	All	All	No
Mockflow	Yes	Yes (visual)	All	Native files	No
Balsamic	No	No	Web app	Images, Sketch, PSD	Yes
Folio	Yes	Yes	Mac	Sketch, PSD, AI	No
Invision app	Yes	Yes	Web app	images, PSD, PDF, AI, Sketch	No
Figma	Yes	Yes(CM)	Web app	All	No
Kaktus	Yes	Yes	Mac	Sketch	No
Plant	Yes	Yes	Mac	Sketch	No
Pics.io	Yes	Yes	Web app	images, audio, video, PSD, PDF, AI, Sketch	No
Abstract	Yes	Yes	Mac	Sketch	No
Trunk	Yes	Yes	Mac	Sketch, PSD, AI	No
Playbook	Yes	Yes(user behaviour)	Web app	PSD	Yes
Versions by Sympli	Yes	Yes	Web app	Sketch	No
GoVisually	Yes	No	Web app	Images, PSD	No
Uspector	Yes	Yes(Visual, UI)	Web app	images	Yes

CM: Commit Message

PSD: Photoshop file type

AI: Adobe Illustration file type

UI: Usability Issues

In summary, Uspector is the only one which is independent from a specific OS, as a web platform, and a set of app-bind file types, such as Sketch, psd etc. This characteristic makes Uspector available to all designers regardless the design tools they use in their workflow for the creation of design assets. Nowadays, all design tools which create art-boards or mockups provide an option to export design assets as images (i.e file formats jpeg, png), but not all of these tools support an option to export as Sketch or psd file formats.

Another difference between *Uspector* and existing tools is the support of design variations. Only our platform, *PlayBook* and Balsamic systems treat design variations of a screen/page as a different entity and not as individual designs. This feature gives to designers the ability to create alternates during the early design phase when exploring concepts and in later stages for reviewers to add their feedback and propose changes. Based on the insights of reviewer's feedback the designer can decide which concept is more "suitable" for a product's needs and so, proceed with the revision only of this design alternative, dropping the other ones, in the later phase of iterative design process. This kind of decisions by the designers are accommodated by Uspector and Balsamic apps, in contrast with the other existing systems, by allowing them to mark a design variation as *Final* or *Official* respectively.

The most important advantage of our platform over the existing ones is the **way** with which a designer is able to compare two versions (diff tool). Tools *Kaktus*, *Plant App*, *Abstract* and *Truck* of the above list are based on MacOS and depend also on the specific design tool Sketch [3]. This tool exports artworks in Sketch file format which allows an external system, such as design version control system, to compare two files and present a rich map of the differences. This means that this subset of systems which support Sketch file format are able to present the visual changes between two design versions to their users. In a different direction tools *Git*, *Folio*, *Invision App* and *Sympli.io* indirectly compare two design versions by allowing user to write relative commit messages on version upload phase. Finally, Playbook (tool 12) follows a totally different approach by comparing interactive prototypes rather than static versions of designs (i.e needed user interaction changed from click to hover events from version x to x+1). Apart from the above methods, Uspector follows a hybrid approach that allow designers to compare two design versions by their commit messages as well as by the set of the usability issues which have been reported after one or more usability evaluations on them.

Usability testing platforms that use the Heuristic Evaluation methodology

Usability testing tools aim to support practitioners and evaluators to conduct and perform usability assessments on a set of design, trying to overcome time bottlenecks due to paper-based activities and face-to-face meetings. There are many ways to classify these tools based on the collaboration between the participants, the selected Usability Evaluation Method (UEM) etc. Some of these categories are

formal and agile, moderated and un-moderated, summative and formative UEM.

In particular, our analysis focused on Usability inspection tools which use the HE methodology. Additionally, our research targeted tools which (i) are platform agnostic and (ii) support usability assessments using HE methodology (extended version with custom heuristics) and (iii) can assist practitioners to asses the usability of their work during multiple phases/stages of product's development process (not only after release/launch of product). Therefore, we concluded on the following set of criteria for systems comparison.

- Platforms
- Usability Evaluation Methodology
- Conduction Stage: Phase of a product's development that a usability test can take place
- Custom heuristics: Support of practitioner's defined set of heuristics

Below, we present briefly each tool as well as their differences with Uspector platform.

1. SUIT
   SUIT [30] is an acronym for Systematic Usability Inspection Tool and it was an Internet-based tool that supports the evaluators during the usability inspection of software applications, using the HE methodology. SUIT makes it possible to reach inspectors everywhere, guiding them in their activities. It has been designed to support the evaluators performing usability inspections, trying to overcome time bottlenecks due to paper-based activities and face-to-face meetings.

   Although its development has been discontinued, SUIT has similar functionalities with Uspector, like inviting evaluators to report, rate the severity and suggest solutions for possible usability issues. Nevertheless, unlike Uspector, it does not support custom heuristics, it refers to the evaluation of systems which have been already launched and it treats HE as an one phase evaluation session without giving the ability to practitioners to filter and aggregate only the set of valid reported issues before evaluators rate their severity.

   One more limitation of SUIT tool is that each reported usability issue refers to a whole page/mockup, lacking the information of the specific area of a design for which the issue has been reported to.

2. Capian
   Capian [15] is a web platform that enables evaluators, while browsing through a website, to save screenshots of it and annotate the specific areas that violate the Nielsen's Heuristics set [44] along with detailed description about the

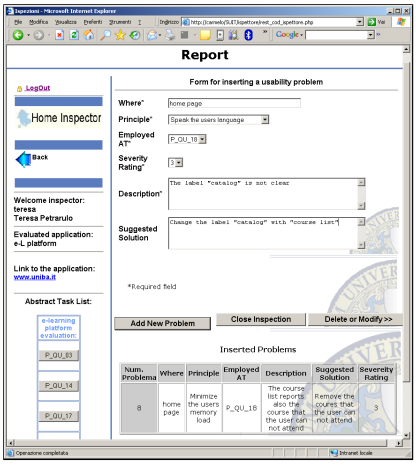


Figure 2.10: Suit tool - Form for inserting a usability problem

violation. Figure 2.11 shows the final HTML report of all reported issues, with the ability for other users to comment on them.

Several limitations of Capian are that it does not support custom heuristics and like SUIT it treats HE methodology as a one-phase process without allowing practitioners to manage their own reported usability issues before they will be rated for their severity.

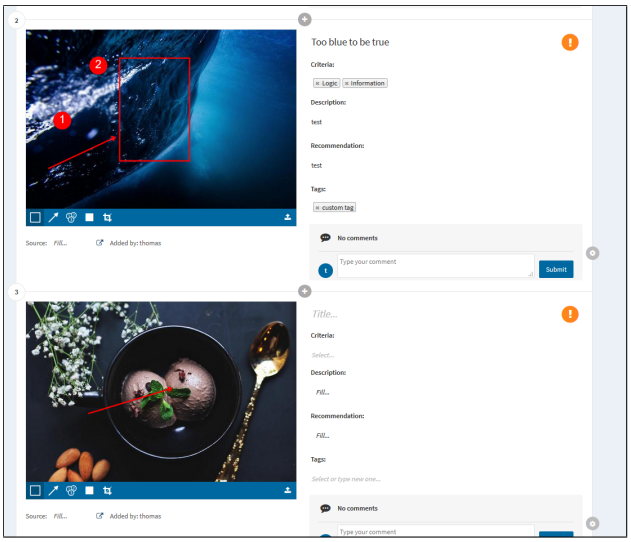


Figure 2.11: Capian tool

3. UXCheck
   UXCheck [17] is a Chrome Extension that helps designers run a heuristic evaluation on any website. UXCheck brings up a list of Nielsen's 10 heuristics

to help an evaluator identify, report and rate issues (figure 2.12).

Although UXCheck supports the definition and use of custom heuristics by a test's practitioner, like SUIT and Capian, it is based on a usability methodology in which the evaluators have to report and evaluate the severity of their own reported usability issues individually at during the report time.

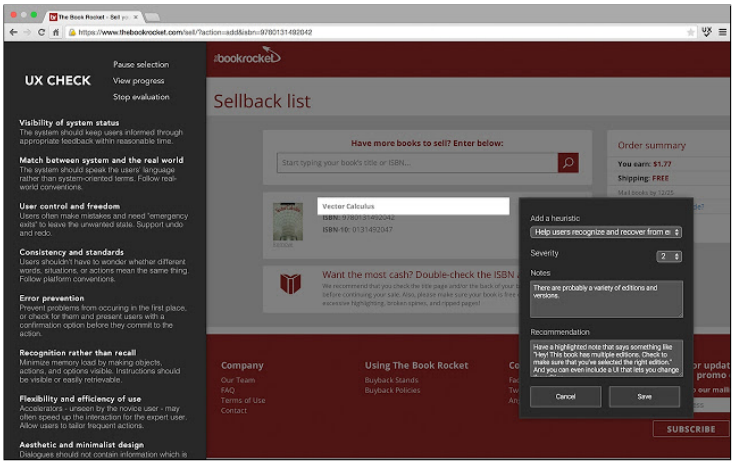
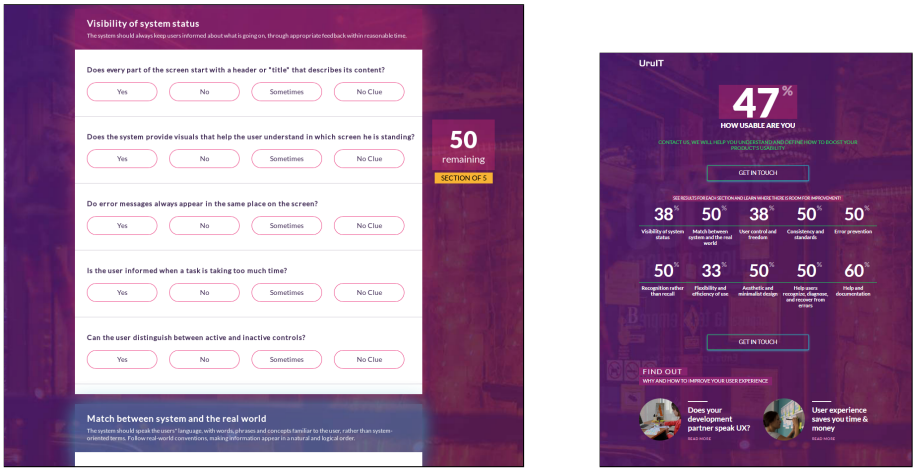


Figure 2.12: Usability issues inspection on websites using UX Check

4. UXQuiz (UruIT)
   UXQui [18] is a web-based quiz that aims to assist designers in evaluating the User Experience (UX) of any digital product which offers a Graphical User Interface (GUI) to its users. The inspected product could be either under design or it could have already launched. Evaluators have to answer a bunch of questions about general usability rules (not Nielsen's heuristics) questions using a predefined rating scale (figures 2.13a and 2.13b).

   A usable feature of UxQuiz is the ability to be used during each phase/stage of a product's development. Still, it does not offer to practitioners the creation, use and management of custom heuristics/guidelines. Also, the evaluators of its usability assessments they have to complete a quiz with heuristics rather than follow the classic HE methodology. That tool's characteristic limit the expression power of evaluators because through the the quiz they cannot freely describe/report a possible usability issue.

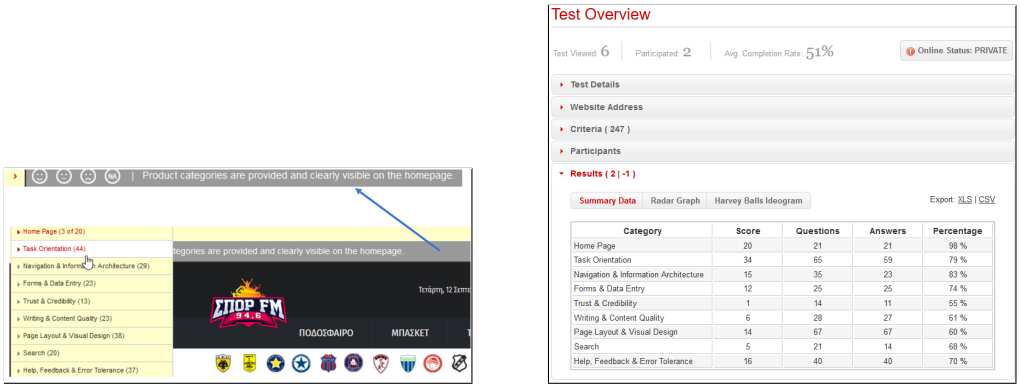
5. Usabilitest
   Usabilitest [50] is a web platform, which consists of a large number of tools for remote usability testing and information architecture like Cards Sorting, Prioritisation Matrix, Heuristic Evaluation, SUS (System Usability Scale). Figures 2.14a and 2.14b show how an evaluator answers a set of predefined questions, which are based on 247 web usability guidelines by Dr. David Travis of Userfocus [38].



(a) Quiz about general usability rules (b) Results page

Figure 2.13: UXQuiz tool

Usabilitest is the more complete solution in comparison with the rest of our review in exissting systems. It offers to a test participants a verbose and complete set of heuristics to use, but it still does not support the creation, use and management of custom heuristics. Moreover, like SUIT and UXQuiz respectively, evaluators report an issue that refers to a whole page/design instead of a specific area of it and the evaluation methodology that is applied to tests is not the classic HE methodology, but rather a quiz with list of questions/heuristics and a 1-5 rating scale.



(a) An evaluator answers questions per system's task by navigating a website (b) Results page

Figure 2.14: Heuristic evaluation tool of Usabilitest platform

An overview of the comparison between Uspector platform and the above existing systems is shown in table 2.2. Each column of the table represents one requirement and each row a usability inspection tool.

Table 2.2: Usability testing tools that use Heuristic Evaluation methodology

	Platform	**UEM**	**Development stage**	**Custom heuristics**
SUIT	Web app	HE(1 phase)	Production	No
Capian	Web app + extension	HE(1 phase)	Production	No
UXCheck	Browser extension	HE(1 phase)	Production	Yes
UXQuiz	Web app	Quiz with heuristics	All	No
Usabilitest	Web app	Quiz with heuristics	Production	No
Uspector	Web app	HE(2 phases)	Design - Production	Yes

UEM: Usability Evaluation Methodology

As Table 2.2 indicates, all existing usability testing tools are web-based and refer to post-launch testing. Therefore, their users (test practitioners) are able to conduct usability tests only on launched systems and not before, during the design phage. Regarding the evaluation methodology, none of them treats HE methodology as a two stage process during which a user firstly examines the UI and individually reports any discovered usability issues, and then rates the **aggregated** list of all reported usability issues. Additionally, our review on existing works shows that non of the them support the creation, use and management of custom heuristics by the practitioner of a usability experiment.

In the final analysis, none the systems in tables 2.1 and 2.2 offers a complete solution for both features, design workflow management and conduction of usability assessments based on HE methodology. In particular, Uspector, as a platform agnostic platform (web app) (i) streamline design workflow by storing design assets (bitmap images), organising them in projects, screens, design variations and versions and (ii) accelerate the validation of new ideas and concepts that design variations and versions carry, through a usability inspection tool which use an extended HE methodology (custom heuristics). Uspector designed and implemented to be used by designers and practitioners both during the design phase (wireframes and high fidelity mockups) and after the launch of a software product (screenshots of a live system).

Chapter 3

System Requirements

This chapter provides the details regarding the environmental and technological components that comprise the Uspector platform. More specifically, it describes the context of use (under which conditions the system should be used), the target group (users with different profiles and roles) as well as the functional requirements for each system's feature.

3.1 Context of Use

3.1.1 Physical Context

Uspector is designed and developed to be a cost effective and accessible platform for designers, usability tests' practitioners and evaluators. aking this into account, there was a need to find the smallest possible intersection among the sets of physical and digital resources that these different user types require in their daily workflows.

More specifically, due to the fact that Uspector is a web platform, the environment in which Uspector is used composed by a typical desktop PC (with mouse and keyboard) or a tablet device, a modern web browser as well as an internet connection.

Terminology in Uspector

This section will describe some of the main entities in Uspector application. It will clarify the terms *Screen*, *Variation* and *Version* as they adopted in the context they are used.

Screens, Variations and Versions

A major logical entity of Uspector platform is the *Screen*. Every screen represents the design of a digital product's logical entity and expresses a specific functionality, feature, or state of this digital product. Designers who use Uspector are expected to have already designed a first version of a screen (using an external design tool)

before uploading it. For example, consider that the product under development is an e-shop; one of its screens would be the "Homepage".

For any screen, multiple variations may exist, with each one expressing a different approach of the desired functionality, feature or state which this specific screen represents. This need is fulfilled by *Variation* entity. Using the above example, the screen for the "Homepage" of the digital product can have two variations, one for dark and one for light theme (color theme) respectively.

Lastly, entity *Version* is the object which describes the outcome of the ID process which a designer follows to design a digital product. Every time a designer refines the design of a screen's variation by addressing a number of its usability issues, a new version of this design is ready to be uploaded in Uspector. For each variation a large number of versions may have been created with the latest of them being the current one. Extending the "e-shop" example, a variation with name "Dark theme" of screen "Homepage" can have three versions, with each one of them to solve a subset of previous' usability issues such as issues about navigation, typography and accessibility.

Designs

In Uspector there is an extensive use of term Design. A design can be either a wireframe (low fidelity) or a mockup (high fidelity) and it represents the work of a designer for a specific version of a screen's variation. Contrary to screen, variation and version, a design does not have abstract meaning, but it refers to the actual artwork of that has been produced via a design tool.

3.1.2 User Context

The main users of Uspector platform are UI(User Interface) designers, practitioners of usability tests and HCI experts who acquire the necessary expertise to evaluate the usability of UI designs using the HE methodology. A registered user of the system can occasionally use the system as any of these roles. For example, the user can firstly use the system as a designer and upload a product's designs. Subsequently, the same user can operate the system as a practitioner by setting up, promoting and conducting a usability test and lastly, Uspector can be used by users who are invited, by tests' practitioners, to evaluate the usability of other designer's work.

Designers

- Main goals: Uspector platform can support designers during an ID process towards developing a product/service/solution. Designers aim at organising and evaluating their designs namely screens, variations and versions. Moreover, they intend to monitor the evolution of their designs' usability

by inspecting in which versions of the design a usability issue reported and resolved respectively.

- Age: There is not a specific age constrain.
- Expertise: Uspector offers variation and version control systems which aim to support the ID process of a digital product. So, basic knowledge of ID process is preferred.
- Experience in using corresponding systems: Experience is not a given and it is not compulsory. In all probability, designers have already used similar version control systems (like Figma, Abstract) during the ID process of a digital product. Moreover, Uspector designed to be easy to learn and equally easy to remember.
- Frequency of use: Frequent. Designers will use Uspector every time a new version of a new or existing design has been created during the (relatively) fast ID process of a digital product.

Practitioners

- Main goals: Practitioners collaborate with other users, who operate as evaluators, aiming to conduct usability tests, following HE methodology, upon designers' work in order to reveal possible usability issues of their designs.
- Age: There is not a specific age constrain.
- Expertise: Knowledge of HE methodology is required. Practitioners need to configure and conduct their usability tests according to HE methodology.
- Experience in using corresponding systems: Experience is not a given and it is not compulsory. In all probability, practitioners have already used online tools to conduct user testing.
- Frequency of use: Occasionally. Practitioners need to use the system whenever they want to create usability evaluation experiments and while such experiments are ongoing. During the conduction of a usability test practitioners should monitor the process and control evaluation phases when needed.

Evaluators

- Main goals: Evaluators aim to participate on usability tests in which they are invited by practitioners. They want to be able to complete a usability test remotely, asynchronously and spend as less time as possible.
- Age: There is not a specific age constrain.

- Expertise: According to HE methodology the evaluators should be HCI experts who are able to examine the UI and assess its compliance with "heuristics" and/or other recognised usability principles. It is not expected that evaluators will have a deep knowledge-understanding of each product's domain (i.e financial/management, music).
- Experience in using corresponding systems: Experience is not a given and it is not compulsory. Although candidate evaluators of a set of designs should be HCI experts. So, they should acquire significant understanding on HE methodology. Nevertheless, they are not required to have prior experience using web platforms which assist the usability evaluation of a digital product's designs.
- Frequency of use: It varies. They are going to use it only upon invitation on a usability test.

3.2 Features and Functional Requirements

3.2.1 User

1. User Sign in
   Users should have the ability to sign in to the platform using their personal credentials.

   Actors: All registered users
   Functional Requirements

   - Users should provide an email and password to sign in.
   - In case of a successful user sign in (valid combination of the provided e-mail and password) to the system, an authenticated session should be created.
   - In case of an unsuccessful user sign in (e.g. wrong credentials, server error) an appropriate error message should be displayed to the user.

2. User Registration
   Users should be able to register to platform using their personal information, such as their name, email and password.

   Actors: All users
   Functional Requirements

   - Users should provide a username, with a maximum length of 20 characters
   - Users should provide an available (not already occupied by other registered user) email address

- Users should provide and confirm a password with minimum length of 5 characters (each character can be alphanumeric or symbol)
- In case of an error (email is occupied, password length, passwords does not confirmed) an appropriate message should be displayed
- System should provide textual hints to users so that they should be able to provide valid credentials

3. User Logout
   Users should have the ability to log out from the platform at any time.

   Actors: All registered users
   Functional Requirements

   - Users should be able sig out from the Uspector system.
   - System should terminate user's session and so, remove all its temporary data from browser's storage, after a successful user logout.

4. Track and Display User Activity
   Users should be able to view a brief history of their actions (relate to content management) on the system.

   Actors: All registered users
   Functional Requirements

   - Users should be able to view the type and date of users' actions for the following Create, Rename, Update and Delete (CRUD) operations:
     - Create, rename, update and delete a project
     - Create, rename, update and delete a usability test
     - Create, rename, update and delete a screen
     - Create, rename, update and delete a screen variation
     - Create, rename, update and delete a version
     - Create, rename, update and delete a heuristics collection
   - Users should be able to view a small description and the date of each one of their actions.
   - Actions of the user's activity should be sorted by date in a descending order (i.e. newer to older) by default.

5. View and Management of user notifications
   Users should be notified asynchronously by the system after an event, relative to their content, occurred. Notifications' content should contain proper messages according to the type of the event. Users should be able to view and clear their notifications at any time.

Actors: All registered users
Functional Requirements

- Users should be able to read older and newer notifications that they have received.
- Users should be able to view details about the actor, the date and a description of the action for the event that a notification relates to.
- Users should be able to clear/delete old notifications.
- Whenever a user is invited to participate in an evaluation test, a respective notification should be displayed that would allow him/her to accept or reject the invitation.

6. Display Profile Details
Users should have the ability to view a full preview of their profile information at any time. Moreover, they should be able to view a brief preview of other users' profile.

Actors: All registered users
Functional Requirements

- Users should be able to preview their profile information, such as their username, profile picture, links to personal profiles on social networks as well as profession's title and experience.
- Default profile picture of a user should be an auto-generated letter avatar according to their email address.
- Users should be able to inspect a partially preview of other users' profile information, such as their profile picture, username and profession details.

7. Profile and Settings Management
Users should be able to set or modify any of their personal information or system preferences at any time so that they can update their identity and expertise.

Actors: All registered users
Functional Requirements

- Users can update their default profile picture by uploading their personal image files. Users should also be allowed to revert back to default profile picture at any time.
- Users should be able to change their profession, experience level as well as the links to their social media profiles.
- Users should be able to change their password after system validate their authorisation by asking their current active password.

- Users should be allowed to change this default system's behaviour and enable-disable email notifications at any moment.

8. Send Email Notifications
   System should send personalised emails to users' registered emails in order to notify them for an incoming invitation to participate on a usability test.

   Actors: Practitioners
   Functional Requirements

   - Email should contain URL to Uspector platform.
   - Email should contain details about the sender, invitation and usability test.
   - A private URL to usability test should also be included so that email recipient can participate on the evaluation session without having an account to Uspector platform.

9. Display State and Statistics of User Content
   Users should be able to view details and statistics about their content's (projects, screens, variations, versions, usability tests etc) current state .

   Actors: All registered Users
   Functional Requirements

   - Users should be able to view in a glance a brief summary about the current state of their content (projects, screens, usability tests).
   - Alongside with the above information, users should be able to view details about the usability tests in which they are invited as well as details about the their pending invitations from practitioners.

3.2.2 Project

1. Display Projects
   Each user can possess a private set of projects, archived or active.

   Actors: Designers
   Functional Requirements Designers should be able to view the following details about each project:

   - Name and project icon.
   - For each project The design which represents the current (latest) version of the final variation (marked as *Final*) of this cover screen defines this project's thumbnail. In case of an empty project, a placeholder image should be used as the cover image. Otherwise the first screen that has been uploaded screen should be used as the cover image.

- Total number of created screens.

2. Project Management
Users should be able to create, rename and delete a project.

Actors: Designers
Functional Requirements

- Users should be able to mark a project as *Favourite*.
- Users should be able to archive a project, when all of its screens have been marked as *Approved*.
- During a project's creation users should be able to provide a name which is not already occupied by other project as well as a brief description of the project (optional).
- Users should be able to delete a project. System should detects accidental users' actions and ask for confirmation before permanently remove a project.
- Users should be able to rename a project. A valid name should have at least two letters and should be unique (i.e. should be used as the name of another active or archived project).

3. Display Details of a Project
Users should be able to view detailed information about a project.

Actors: Designers
Functional Requirements
For a project the following details should be displayed:

- Name
- Icon and domain's information of the digital product, which the designs describe to.
- Specific indicator if the project is marked as Favourite.
- Name of the cover screen.
- Total number of screens categorised by state (*In progress*, *Under evaluation*, *Approved*).
- Creation date and time.
- Total number of usability tests which are conducted for this project.

4. Projects Collection Management
Users should be able to search, sort and filter their projects according to a set attributes.

Actors: Designers
Functional Requirements

- Users should be able to search their projects by name. System should retrieve and display search results during the query construction by designers. Moreover, system should present the number of retrieved results to users.
- Users should be able to clear current search query.
- All possible sorting options should be the following:
  - By name (ascending, descending)
  - By creation date (new vs. old)
  - By number of screens
- Users should be able to filter their projects based on the following attributes:
  - Activeness status (active, archive)
  - Favourite status

5. Display Workflow of a Project
   Each project consists of a set of screens. System should display which of them are in progress i.e. screen needs testing until it is ready), under evaluation (i.e. when belonging to at least one usability test) or Approved.

   Actors: Designers
   Functional Requirements

   - Users should be able to view their list of screens categorised by their progress status (*In progress*, *Under evaluation*, *Approved*).
   - For each status category the following details should be displayed:
     - In progress: Name of final screen's variation and name of its current version
     - Under evaluation: Number of usability tests on which this screen is evaluated.
     - Approved: The full date when designer marked screen as *Approved*.

6. Display and Update Project Details
   Users should be able to set and/or update its details about their projects.

   Actors: Designers
   Functional Requirements

   - Users should be able to set and update the following details about the digital product which project represents:
     - Brief project's description with length up to 200 characters.
     - Project icon (png, jpg formats).
     - Brief digital product's domain description with length up to 200 characters.

3.2.3 Screen

1. Display Screens of a Project
   Users should be able to view all screens of a project.

   Actors: Designers
   Functional Requirements
   Users should be able to view the following details for each screen:

   - Name and current state (*In progress*, *Under evaluation*, *Approved*).
   - Thumbnail of a design which represents the current version of the Final screen's variation.
   - Name of Final screen's variation.
   - Name of current version of Final screen's variation.
   - Total number of unresolved usability issues. These issues represent the unresolved issues of the current version of Final screen's variation
   - Total number of screen's variation .
   - A screen should properly indicated whether it's a newly uploaded one or not.

2. Screen Management
   Users should be able to create, delete, rename a screen. Moreover, users should be able (un)mark a screen as Approved.

   Actors: Designers
   Functional Requirements

   - Users should be able to upload one or more images from their local storage in order to create one or more screens respectively.
   - For every newly created screen system should do the following operations:
     - Initialise the name of the screen using the filename without its file extension. For example, if a user uploads a file with name "Homepage.jpg" then screen's name should be "Homepage"
     - In case of an already occupied name, the system should initialise the screen's name by appending a number to the filename that would indicate how many screen have the same name increased by 1 (one).
   - Owners of screens should be allowed to rename them. System should not accept empty or occupied names.
   - Users have the ability to delete on one or more screens. Regardless the number of selected screens, users should confirm their intention before any screens are removed permanently.

- System should automatically delete all variations and versions of a screen which is going to be deleted.
- Users should not be able to delete a screen if its status is *Under Evaluation* as a result of its participation on one or more usability tests.

3. Screens Collection Management
Users should be able to search, sort and filter screens of their projects. Moreover, users should be able to change the screens' view mode according to the device type for which they have been designed.

Actors: Designers
Functional Requirements

- Users should be able to search by name their screens.
- Users should be able to clear current search query.
- Users should be able to sort the screen by (in their following order):
  - Name (ascending, descending)
  - Creation date and time (new, old)
  - Total number of unresolved usability issues
- Users should be able to filter their projects based on their status (*In progress*, *Under evaluation*, *Approved*).
- Users should be able to adjust the view mode of a Screen element in the UI. More specifically, system should provide options to users to adjust the Screen elements of a project according to the type of device which have been designed for (desktop, mobile).

3.2.4 Screen Variation

1. Display Variations of a screen
Every designer should be allowed to view the set of variations of any of his/her screens.

Actors: Designers
Functional Requirements
For each screen's variation the following details should be displayed:

- Name (i.e Variation D)
- Number of its current version
- Variation's status (i.e. whether it is marked as Final or not)

2. Screen Variation Management
Users should be able to create, rename or delete a screen's variation.

Actors: Designers
Functional Requirements

- Users should be able to create a screen's variation by optionally defining a custom name and a commit message which describes the differences of the new variation against the existing ones.
- The default name of a new variation composed as a string "Variation X", where X is a variable which takes sequential capital letters in the English alphabet. For example, the first variation is called "Variation A", the second "Variation B" and so on.
- System should automatically create the first version of a newly created screen's variation with the name "Version 1" and commit message "Initial Message".
- System should allow users to replace the file before the creation of a screen's variation to prevent errors.
- Users should be able to rename an existing screen's variation
- In case of empty or already occupied names system should display proper error messages.
- Users should be able to delete a screen's variation. If the variation is part of a usability test then the deletion is prohibited by the system
- When the user deletes a screen variation, all its versions should be deleted as well.

3. Display Details of a Screen Variation
Users should be able to view details about a variation of a screen.

Actors: Designers
Functional Requirements
System should display the following details for a screen's variation:

- Name (i.e Variation B)
- Specific indicator only for variations which are marked as Final by their owners
- Commit message, explaining the differences with already existing screen's variations
- Creation date and time

4. Mark a Screen Variation as *Final*
Users should be able to set a screen's variation as the final one. This mark express the decision of the designer to keep working with the direction that this screen's variation represents.

Actors: Designers
Functional Requirements

- By default, the first uploaded variation of a screen is marked *Final* by the system. If a final variation deleted the next variation by creation date is marked as *Final*.
- Only one variation of a screen should have been marked by its owner as *Final* at any time.

3.2.5 Version of a Screen Variation

1. Display Versions of a Screen Variation *Every designer should be allowed to view the set of versions for a specific screen variation.*

   Actors: Designers
   Functional Requirements
   For each version the following details should be displayed:

   - Name (i.e Version 2)
   - Total number of (un)resolved usability issues

2. Version Management
   Users should be able to view, create, rename and delete a version.

   Actors: Designers
   Functional Requirements

   - Users should be able to provide name on version creation. Default name should be "Version X", where X is a variable which takes sequential numbers starting from 1. For example, the first version is called "Version 1", the second "Version 2" and so on.
   - Users should be able to provide a commit message which explains the differences between the new version and the previous versions.
   - Users should be able to delete a version after they confirm their intention.

3. Display and Manage usability issues of a version
   Users should be able to view and resolve any pending usability issues of a version. In case a version already contains a list of "resolved" issues, the user should be able to mark any of them as Unresolved.

   Actors: Designers
   Functional Requirements

   - Users should be able to view the total number of (un)resolved usability issues, which have been reported by evaluators during tests, of a design's version.

- For each usability issue the system should display its (i) title (ii) violated set of heuristics (iii) description (iv) average severity ratings (as given by the evaluators) and (v) suggested solutions.
- Designers should be able to resolve a usability issue on a specific version, after confirming their intention.

3.2.6 Heuristics Collection

1. Display and Management of Heuristics Collections
   Users should be able to view the list of default and custom heuristics collections.

   Actors: Practitioners
   Functional Requirements

   - For each heuristics collection the following details should be displayed:
     - Name (i.e Heuristics for Mobile apps)
     - Total number of heuristics rules
     - Accessibility status (Protected or not)
   - Protected heuristics collections should not be edited or deleted by users.
   - Users should be able to rename a non protected heuristics collection. Empty or already occupied names should not be allowed and system should display proper error messages.
   - Users should be able to delete a non protected heuristics collection after confirming their intention.

2. Display and Manage Heuristics of a Heuristics Collection
   Users should be able to view the list of heuristics of a heuristics collection.

   Actors: Practitioners
   Functional Requirements

   - For each heuristic the following details should be displayed:
     - Title (i.e Visibility of system status)
     - Description with a maximum length of 200 characters
   - Users should be able to rename a heuristic
   - Users should be able to delete a heuristic after asking them for confirmation
   - Rename and delete actions should be prohibited to heuristics which belongs to a protected heuristics collection

3.2.7 Usability test

1. Display Usability tests
   Users can view a set of usability tests classified by the role of them on each test. These roles can be either the practitioner who manages the test or one of the evaluators who participate in it.

   Actors: Designers
   Functional Requirements
   Users should be able to view the following details for each usability test:

   - Name
   - Availability status, locked or unlocked. (**Only for Practitioners**)
   - Evaluation progress status which is indicated by usability test's states. These states are the following: *Idle* (Not started yet, paused/locked), *Phase 1*, *Phase 2* and *Completed*. These states have been defined based on HE methodology.
   - Deadline of current evaluation progress status.
   - Name of the project for which the test is conducted.
   - Total number of designs which are going to be inspected by test's participants (evaluators).
   - A snippet of the invited evaluators who participate on the test. This snippet have to contains information for up to five evaluators. For each evaluator, system should present the profile picture accompanied by the username. (**Only Practitioners**)
   - A proper message with the total number of invitations with status *Pending* (evaluator has not answered yet).

2. Usability test Management
   Users should be able to create, rename and delete a usability test.

   Actors: Practitioners
   Functional Requirements

   - For each new usability test users should provide a, non already occupied, name and select an existing project of designs and choose the evaluation methodology (HE, User testing) on which the new usability test will be based.
   - Practitioners should be able to delete their usability tests after confirming their intention.
   - Practitioners should be allowed to change usability tests' names. System should not accept empty or already occupied names.

3. Usability Tests Collection Management
Users should be able to search, sort and filter their usability tests based on a set of attributes.

Actors: Practitioners, Evaluators
Functional Requirements

- Users should be able to search by name their personal and shared usability tests. Moreover, system should present the number of retrieved results to users.
- Users should be able to view the original list of usability tests by clearing any filtering options (e.g. search, filters, sorting).
- A set of usability tests can be sorted by:
  - By name (ascending, descending)
  - By creation date (first, last). Users should be able to sort by this attribute only for the set of usability tests on which user is the practitioner.
- Users should be able to filter their projects based on the following attributes:
  - User role in tests (practitioner, evaluator)

4. Answer to an Invitation for Participation on a Usability Test
Invited users to a usability test should be able to accept or decline the invitation before participating to it.

Actors: Practitioners, Evaluators
Functional Requirements

- Users should be able to accept or decline an invitation before gaining access on the test. Access should be denied until the invited user reply to the invitation.
- System should display the following details about the invitation:
  - Practitioner's username and email.
  - Invitation date from the practitioner of the test.
  - Usability test details such as project name, domain information as well as the scenarios of use.
  - In case an evaluator rejects an invitation, the system should archive the invitation and prevent the access to this usability test.
  - In case an evaluator accepts an invitation, the system should grant access to usability test in order to start the evaluation session.

5. Display Summary and Statistics about a Usability Test
   Users should be able to preview a brief summary of their usability tests' details.

   Actors: Practitioners
   Functional Requirements

   - System should display an overview of usability test's current state. More specifically, the following details should be displayed:
     - Availability (public, private)
     - Evaluation session status (phase 1 or 2)
     - Deadline of current phase
     - Total number of designs
     - Total number of heuristics collections
     - Progress of each evaluator (i.e user 1 completed *Phase 1* of the evaluation)

3.2.7.1 Usability Test: Monitoring and Participation

1. Display set of reported Usability Issues
   All participants (practitioner and evaluators) of a usability test should be able to view a subset of the reported usability issues during each phase of a evaluation session.

   Actors: Practitioners, Evaluators
   Functional Requirements

   - Practitioners of a usability test should be able to view the whole set of reported usability issues which exposed during *Phase 1* of an evaluation session by the evaluators.
   - During *Phase 2*, evaluators have to rate the severity of each usability issue which is approved by the practitioner after the end of *Phase 1*.
   - Each usability issue should contain the following details:
     - Details about the author: Practitioners should see all details about the author of usability issues, such as profile photo, name and email.
     - Title, description, set of violated heuristics, severity rating and suggested solution to the problem.
     - Date of the report by its author (evaluator).

2. Update of a Usability test's Visibility Status
   Practitioners of a usability test should be able to (un)lock it during the evaluation session.

Actors: Practitioners
Functional Requirements

- Practitioners should be able to (un)lock a usability test only when evaluation session is on phases 1 or 2. In case a test is not started yet or it has already completed then it is locked, and so not available to evaluators. In these conditions the test is state *Idle*.
- System should automatically add test's settings in view mode only in case a usability test is unlocked. Practitioners should be able to configure a usability test when it is in *Idle* state, which means it has either not started yet or its practitioner has temporally locked it.

3. Control of Evaluation Session State
Practitioners should be able to control the current state of evaluation process for a specific usability test. Those different states of the process are Not started, Phase 1, Phase 2 and Completed.

Actors: Practitioners, Evaluators
Functional Requirements

- Practitioners should not be allowed to start or complete phases 1 and 2 if the previous phase is not completed. More specifically, there are some requirements for each of the following transitions:
  - *Idle* to *Phase 1*: Settings of usability test should be completed.
  - *Phase 1* to *Phase 2*: Settings of usability test should be completed and *Phase 1* should be completed.
- Practitioners should be able to inspect all evaluators' progress (current phase of evaluation process) at any time during the evaluation session which they participate.
- Evaluators should be able to proceed to phase 2 as soon as they have inspected the usability of every available design in a usability test.
- Evaluators should be able to complete a usability test after they rate the severity of all reported usability issues during Phase 1. Moreover, the evaluators should confirm their action to complete the test before system save their ratings on the usability test.
- Practitioners should confirm any action which is related with update of the evaluation progress' status for any evaluation test.

4. Start an evaluation session of a usability test
Evaluators should view a proper welcome message alongside with test's details and information of the evaluation session.

Actors: Evaluators
Functional Requirements

- System should display information about the usability test such as:
  - Name
  - Deadlines for *Phase 1* and *Phase 2*
  - Number of designs to be inspected by the evaluators
  - Details about the practitioner, such as username and picture profile
  - Number of heuristics
- System should display a brief description about the process of the evaluation session and the methodology that the evaluators should follow in order to complete the test (phases, steps and a small tutorial).

5. Change Theme of the Design/Mockup Viewer
   All users of Uspector (independent of their role) can customise the page of design viewer between a light, a grey and a dark theme. Each one of these themes apply a different color palette to all visual elements of this page.

   Actors: Practitioners, Evaluators
   Functional Requirements

   - Each theme should provide high contrast between visual elements of the page.
   - Users should be able to update their preferences and alternate themes at any time.

6. Display Details of a Usability Test
   Users should be able to view details about the process and methodology of the evaluation session.

   Actors: Practitioners, Evaluators
   Functional Requirements

   - System should display information about the usability test such as:
     - Name
     - Deadlines of each phase as defined by the practitioner during test's configuration
     - Status of evaluation session
     - Project name (**Only Practitioners**)
     - Number of accepted and pending invitations to evaluators (**Only Practitioners**)
     - Creation Date (**Only Practitioners**)
     - Number of designs to be inspected
     - Details about test's practitioner, such as username and picture profile (**Only Evaluators**)

  - Number of heuristics
- Practitioners and evaluators, who have already accepted their invitations to a usability test, should be able to view the current status of its evaluation process.
- Users should be able to view the title and a brief description (max 200 characters) about a heuristic/rule. System should display provide the ability to a test's participants to recall the definition of each avaailable heuristic at any time during an evaluation session.

7. Display set of Designs for inspection
   The Practitioner and the evaluators of a usability test should be able to view the set of designs which are selected during the test configuration and they are evaluated during the current evaluation session.

   Actors: Practitioners, Evaluators
   Functional Requirements

   - System should display the name and the thumbnail of usability test's designs.
   - At any time, participants should be able to indicate the active design among the others.

   [Here]

8. Search and Navigation among Designs for Monitoring and Evaluation
   Evaluators should be able to search and quickly navigate among the designs of a usability test in order to find and inspect a specific design.

   Actors: Practitioners, Evaluators
   Functional Requirements

   - System should support live searching on a usability test's available designs. As the user constructs a query, system should filter the designs by name, and presents to the user the filtered set as well as the total number of them.
   - Users should be able to indicate which design is the active one (loaded in the viewer), during their navigation to the set of available designs of a usability test.
   - Practitioners and evaluators should be able to fast and intuitively navigate and load in the viewer whichever design is available in a usability test.

9. Usability Issue Report, Edit and Delete
   During Phase 1 of an evaluation session evaluators can report, edit and delete

usability issues on specific positions/points of a design.

Actors: Evaluators
Functional Requirements

- Each newly reported usability issue should contain the following information:
  - Position (X, Y) upon design, where X, Y are the horizontal and vertical coordinates respectively.
  - Title of usability test.
  - Set of criteria/heuristics which reported usability issues violates.
  - A brief description (max 200 characters) of the reason of a usability issue report.
- Reported usability issues should be displayed correctly on any of the supported screens.

10. Usability Issues Rating
During Phase 2 of an evaluation session, evaluators should rate the severity of all reported usability issues of each design. A reported usability issue can be created from any of the evaluators during Phase 1 of evaluation session. Moreover, system should allow evaluators, optionally, to suggest solutions to the practitioner on how to resolve one or more of the discovered usability issues.

Actors: Evaluators
Functional Requirements

- Evaluators should rate the severity of all reported usability issues during a usability test. The provided scale to evaluators is the severity rating scale which defined by J. Nielsen [46] [44].
- System should indicate to each evaluator which usability issues have already been rated and for which this process is still pending.
- Evaluators should be able to suggest a solution to the practitioner about a usability issue that they report on a design.
- Evaluators should be able to edit a severity rating after its submission but before they complete the phase 2 of the evaluation.

3.2.7.2 Usability test Configuration

(a) Display and Configuration of a Test's Settings
System should display the current configuration settings of a usability test at any time during its lifetime (before, during and after an evaluation session takes place)

Actors: Practitioners
Functional Requirements

- Settings should be locked in case a usability test's state is *Phase 1* or *Phase 2* which means that evaluators are able to report and rate usability issues. Practitioners should be allowed to configure one or more test's settings only if they manually lock it temporarily and so put the test in the state *Idle*.
- System should notify (using proper warning messages) the practitioner of a test if settings are not available for configuration (state is not *Idle*).

[Here]

(b) Configuration of a test's General Settings
Every usability test should have a set of general settings.)

Actors: Practitioners
Functional Requirements

- The following options for configurations should be offered:
  - Deadlines for each evaluation phase (*Phases 1, Phase2*). System should not allow practitioners to define a deadline for *Phase 1* with a date older that the current one. In the same direction, the deadline for phase 2 should be ahead of the specified deadline for phase 1. Both deadlines should be required by the test's practitioner before a test starts.

(c) Management of available Heuristics Collections in a Test
Practitioners should be able to define which heuristics collections will be used in a usability test. These heuristics collections should be available for use to evaluators during Phase 1 of evaluation in order to assist them to report and document possible usability issues for the usability test's designs.)

Actors: Practitioners
Functional Requirements

- A practitioner should be able to select a subset of available heuristics collections which are going to be used by evaluators in a usability test.
- Search all heuristics collections and display the number of search results.
- Practitioners should be able to (de)select a single or all heuristics collections.

(d) Include/Exclude Designs in a Usability Test
Each usability test is connected with a project which consists a set of

screens. Practitioners should be able to include a subset of designs (specific variation and version) which need to be evaluated. These designs will be available for inspection to evaluators during the evaluation session.

Actors: Practitioners
Functional Requirements

- The system should display all screens of the project. For each screen system should display a screen's name, selected variation and version as well as a thumbnail of it.
- A practitioner should be able to include and/or exclude a design from the selected set of designs which will take part in the evaluation process of a usability test.
- Search all designs and display the number of search results.
- Practitioners should be able to select, deselect or leave deselected a single or all designs.
- Practitioners should be able to change selected variation and version of a specific screen before they include it to a usability test. System should display details for each variation and version helping a practitioner to select with ones of them to include to a usability test for evaluation .

(e) Management of Invitations in a Usability Test
Each usability test is connected with a project which consists a set of screens. Practitioners should be able to include a subset of designs (specific variation and version) which need to be evaluated. These designs will be available for inspection to evaluators during the evaluation session.

Actors: Practitioners
Functional Requirements

- A practitioner of a usability test should be able to search users using their emails. System should retrieve results only if query length is more than two characters.
- For each retrieved search result system should display user's profile picture and name
- A practitioner should be able to invite someone who has not account on the Uspector platform using his/her email
- System should allow practitioners to invite multiple users
- System should display all invitations of a usability test. For each invitation, details such as user's email and profile picture as well as the date of invitation should be displayed.
- Practitioners should be able to search invitations by user's email

- System should provide a unique share URL for each invited evaluator which is not a registered user of the platform. This URL gives access to the evaluation session of the usability test.
- Practitioners should be able to cancel an invitation that has been sent to candidate evaluators. In that case, an invited user will still be able to answer, as soon as they acquire the invitation's email, but he/she will not be able to use the provided unique URL to the test because it will not be valid anymore.
- System should allow to enable and/or disable the access to a usability test, using a share URL, for a specific invitation.

Chapter 4

System Architecture

This chapter presents (i) an abstract overview of system's architecture, (ii) a conceptual data model of the system and (iii) the core technologies which were used to develop Uspector.

In particular, it describes the overall system architecture (i.e. the main components and their relations), the data model (i.e. entities and their relations), as well as the different technologies used to build the front and back-end components of the platform.

4.1 System Architecture

Overview

Uspector is a web platform and its architecture follow the client-server model. That means that the server hosts, delivers and manages most of the resources and services to be consumed by the clients.

The high-level architecture of Uspector platform has been driven by the aforementioned features and functional requirements which are described in chapter 3. It consists of several interconnected components with specific role in the functionality of the platform.

Front-end Architecture

A high-level scope of client-side part's architecture is showed in figure 4.1). The main module is called *App* and consists of two sub-modules *Account*, *Home* as well as a collection of utilities modules. Utilities contribute to track users' actions, manage push notifications, enhance intercommunication among view's components (each page/view compose a component), support the communication with the RESTful API and manage data store/retrieval operations from/to browser's local storage.

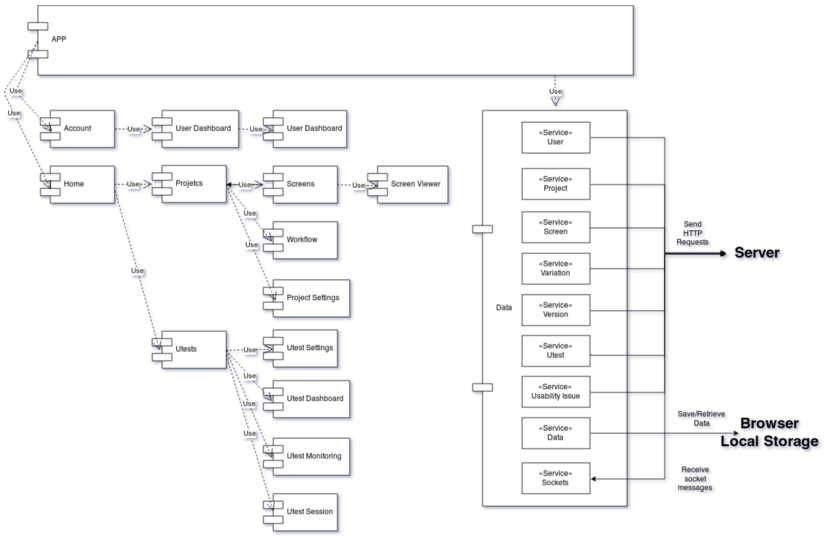


Figure 4.1: Front-end architecture

Account Module

The *Account* module contains the modules and components which are responsible to manage the visible views of a registered user. More specifically, it represents the private section of application, which provides the workspace, the tools and the information to the user to manage and/or evaluate the designs of other users. Module *Account* contains the following sub-modules:

- User Dashboard
- Projects
- Screens
- Workflow
- Project Settings
- Screen Viewer
- Usability Tests
- Usability Test Dashboard
- Usability Test Configuration
- Usability Test Session
- Heuristics Collections

Home Module

The *Home* module contains the modules and components which are responsible to manage the public views (Homepage with authentication forms) of Uspector platform. Therefore, it is the entry gate (app's public side) of Uspector platform for the incoming visitor and so, its role is to manage the task of user authentication in the client-side of the application, in terms of asking users' credentials, showing proper messages on unsuccessful user attempts and navigating users in the rest system on a successful sign in.

Data Module

The most used modules of Utilities is the *Data* module which is used for the management of browser's local storage, communication among different components and the communication with the RESTful API. More specifically, it contains the following modules:

- User Service: Implements CRUD operations for the data type *User*
- Project Service: Implements CRUD operations for the data type *Project*
- Screen Service: Implements CRUD operations for the data type *Screen*
- Variation Service: Implements CRUD operations for the data type *Variation*
- Version Service: Implements CRUD operations for the data type *Version*
- Usability issue Service: Implements CRUD operations for the data type *Usability issue*
- Usability test Service: Implements CRUD operations for the data type *Usability test*
- Data Service: Manage local storage of browser. This service focus on save and retrieve all data which are related logged in user's session, such as token, preferences and settings.

Back-end Architecture

The business logic of system is managed by its nodejs server. The main building blocks of the back-end architecture are: (i) Server app, (ii) Services endpoints (RESTfull API), (iii) Web socket emitters and (iv) Data models component. As shown in figure 4.2, the server app is responsible to initialise the server and manage incoming requests to assets (images, illustrations etc), the endpoints of services manage the incoming http requests for them, the web sockets emitters guarantee asynchronous communication between a client and the server and lastly the Data Models Component is responsible to model and manage the stored data in the database.

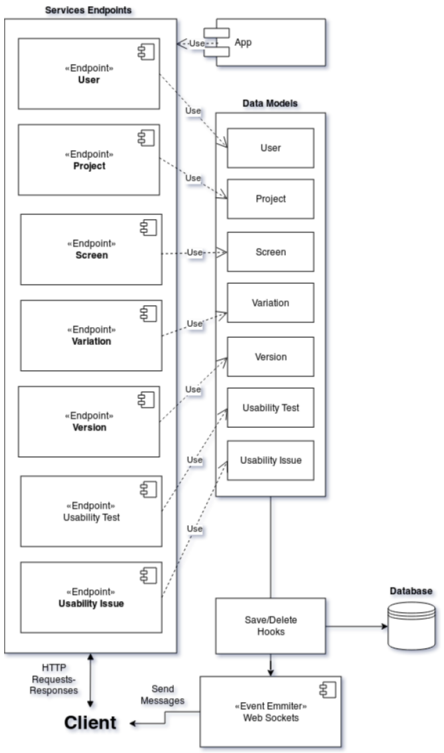


Figure 4.2: Back-end architecture

4.2 Data Modelling

4.2.1 Overview

A critical task for the development of Uspector was to define a conceptual data model that describes a map of concepts as well as their relationships used for the design and development of the database. The definition of this map is directly connected with the information that is stored in the database.

For the purposes of the design and development of Uspector's database we used a document-based database, namely MongoDB [12], due to the scalability and the freedom that offers in modelling data and defining relationships among them. Data schemas do not need to have an immutable structure beforehand, and so, they can be modified on the fly according to applications needs. Lastly, a Mongo database contains collections which consecutively contain documents, the data of which, can be retrieved easily in JSON format.

MongoDB does not support joins (like SQL databases) and so, it allowed us to model data either with embedded documents (denormalised model) or referencing documents (normalised model) (figure 4.3) according to application's needs.

In Uspector we use a hybrid data model that combines embedding and referencing models. Two key factors defined the design of data model, the **frequency**

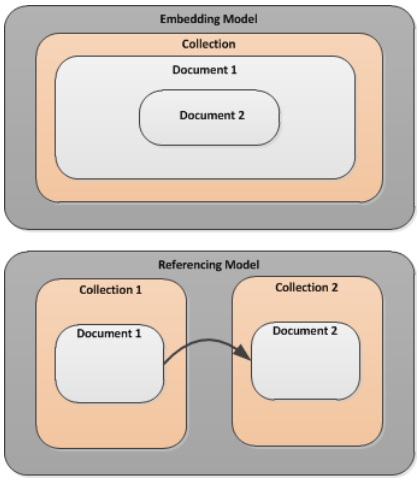


Figure 4.3: Referencing and Embedding data models [2]

of read-write operations and the **growth** of documents. Small models, such as *Notifications* and *HeuristicsCollection*, defined as embedded to bigger ones, like *User* and *Utest* respectively. On the contrary, models such as *Project*, *Screen* and *UsabillityIssue* which require frequent and fast CRUD operations need to be independent (not embedded) models. That decisions lead to a scalable and dynamic data model which offers fast CRUD operations, but at the same time, they made the preservation of atomicity for complex operations (operations which relate more than one model) a difficult challenge for us.

In particular, architecture of our database consist of the following building blocks:

- Collections: User, Project, Screen, Variation, Version, UsabilityIssue, UsabilityTest, Sessions
- Documents: Instances of each collection i.e for a registered user "George" there is a document of collection User stored in the database
- Fields: Attributes of each collection
- References: A connection between two collections

4.2.2 Uspector entities

After the collection of functional and non functional requirements, the basic logical entities are extracted. An overview of Uspector's database schema is shown in class diagram which is shown in figure 4.4.

Figure 4.4: The conceptual data model of Uspector platform

User

Entity *User* describes the end-user of the system. Users can be designers, practitioners, evaluators or any combination of them. Uspector offers functionalities for both registered and unregistered end-users. The fields of the User entity are described in Table 4.1.

Table 4.1: User entity

Field	**Type**	**Description**
shortId	String, unique	A short id which uniquely identify a user
username	String, required	Name of user
email	String, unique, required	Email of user
role	String	Role of user

avatar	String	Path to profile picture of user
password	String, required	Password of user
personal		Personal information of user
activity	ObjectId[User]	History of user's actions
project	ObjectId[Project]	List of reference to user's projects
heuristicsSets	[SubDocument]	List of heuristics collections
notifications	SubDocument	Notifications preferences
salt	String	Salt of user's password
utests	ObjectId[Utest]	List of references to user's usability tests

Project

Entity *Project* represents a folder of screens. A project can consist of multiple screens and it can be associated with multiple usability tests. Each one of them can be responsible to evaluate a specific design of a subset of project's screens. More details about the fields and associations of the model *Project* are described in Table 4.2.

Table 4.2: Project entity

Field	**Type**	**Description**
shortId	String, unique	A short id which uniquely identify a project
name	String, required	Name of project
description	String	Brief description of project
domain	String	Product's domain description
archived	boolean	Status of project (active, archived)
favorite	boolean	Indication if a project is favorite
appIcon	String	Icon of product
owner	ObjectId[User]	Reference to project's owner
screens	ObjectId[Screen]	List of references to project's screens
coverScreen	ObjectId[Screen]	Reference to cover screen of a project

Screen, Variation, Version

These three entities, which are defined in subsection 3.1.1, describe (i) the design of a product's logical entity which express a specific functionality, feature, state or

component of this product, (ii) a different approach of the functionality, feature or component which is represented by a screen and (iii) an instance of the iterative design process for a screen variation, respectively.

A screen can consists of multiple variations and each variation can consist of multiple versions. More details about the fields and associations of these models are shown below, on tables 4.3, 4.4 and 4.5.

Table 4.3: Screen entity

Field	**Type**	**Description**
shortId	String, unique	A short id which uniquely identify a screen
name	String, required	Name of screen
status	String	Progress status of screen (in-progress, under-evaluation, ready)
owner	ObjectId[User]	Reference to owner of screen
project	ObjectId[Project]	Reference to ancestor project
variation	ObjectId[Variation]	List of references to screen's variations

Table 4.4: Variation entity

Field	**Type**	**Description**
shortId	String, unique	A short id which uniquely identify a variation
name	String, required	Name of variation
final	boolean, required	Indicates if a screen's variation is the final one
commitMessage	String	Message which describes the diff with other variations of screen
owner	ObjectId[User]	Reference to variation's owner
versions	ObjectId[Version]	List of references to variation's versions
project	ObjectId[Project]	Reference to ancestor project
screen	ObjectId[Screen]	Reference to ancestor screen

Table 4.5: Version entity

Field	Type	Description
shortId	String, unique	A short id which uniquely identify a version
name	String, required	Name of version
commitMessage	String, required	Message which describes the differences against previous version
imgPath	String	Path to file(image) of design
originalName	String, required	Name of file in the client's storage
systemName	String, unique, required	Name of file in the server's storage
current	Boolean, required	Indicates if a version is the current one
usabilityIssues	ObjectId	List of references to usability issues
owner	ObjectId[User]	Reference to owner of version
project	ObjectId[Project]	Reference to ancestor project
screen	ObjectId[Screen]	Reference to ancestor screen
variation	ObjectId[Variation]	Reference to ancestor variation
utests	ObjectId[Utest]	Reference to ancestor usability tests

Usability issue

Entity *Usability issue* hold details about a usability problem reported by an evaluator in a specific design/mockup. For example it can contains information about the set of heuristics that this design violates, a relative description and a rating of the reported problem. More details about the fields and associations of the model *Usability issue* are described in Table 4.6.

Usability test

Entity *Usability test* describes an experiment for the evaluation of designs' usability. It is the link between a designer and an evaluator, because through the evaluation session the participants (evaluators) inspect a designer's work in order to report possible usability problems. A practitioner can have multiple usability tests and each one of them can consists of multiple designs and associate with multiple evaluators. More details about the fields and associations of the model *Usability test* are described in Table 4.7.

Table 4.6: Usability Issue entity

Field	Type	Description
shortId	String, unique	A short id which uniquely identify an issue
orderId	Number	Composite id based on user, design and usability test
posX	Number	Horizontal coordinate of position
posY	Number	Vertical coordinate of position
owner	ObjectId[User]	Reference to the creator of a usability issue
project	ObjectId[Project]	Reference to the project in which the design belongs
screen	ObjectId[Screen]	Reference to the screen in which the design belongs
variation	ObjectId[Variation]	Reference to the screen's variation in which the design belongs
utest	ObjectId[Utest]	Reference during which the issues has been reported
version	ObjectId[Version]	Reference to the screen's version in which the design belongs
title	String	Short title of usability issue
heuristics	Array of heuristics	Set of violated heuristics
description	String	Description of usability issue
suggestedSolution	String	Suggested solution which solves usability issue
severity	Number	Mean Severity rating of usability issue

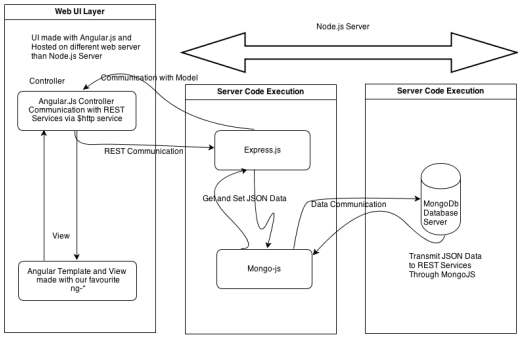
Table 4.7: Usability Test entity

Field	Type	Description
shortId	String, unique	A short id which uniquely identify an issue
name	String	Name of test
practitioner	ObjectId[User], required	Organizer of the test
type	String	Type of evaluation methodology
status		Availability and progress of evaluation session
project	ObjectId[Project]	Reference to the project in which the design belongs
scenarios	List[String]	Two scenarios of use
deadline	SubDocument	Deadlines of evaluation phase 1 and 2
severityRatingScale	SubDocument	Rating scale for the severity of a usability issue
heuristicsSets	[SubDocument]	List of heuristics collections
invitations	[SubDocument]	
designs	ObjectId[Version]	Reference to version of a screen's variation

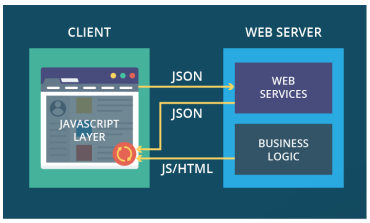
4.3 Technologies

The Uspector platform is designed and developed based on MEAN stack model (figure 4.5a). MEAN is a free and open-source JavaScript software stack for building dynamic, fast and robust web applications. MEAN comprises of a set of four technologies stands for: (M) MongoDB [12], the database that will be use, (E) Express.js [4], (A) AngularJs [1], the client-side technology which is a JavaScript library to render the application, and finally (N) Node.js [39], which is the JavaScript server-side language used for the server logic. The development of Uspector based on the Angular-Fullstack Generator [8] which is a boilerplate code generator. That decision has led to the fact that codebase has been developed following the best software patterns and techniques (i.e authentication, data consistency etc), both at front-end and back-end.

Based on the MEAN stack, the following tools and technologies have been used for the development of Uspector platform. The *Overall architecture cycle* in Uspector platform is illustrated in figure 4.6.



(a) MEAN stack model [36]



(b) Single Page Application architecture

Figure 4.5: Uspector overall architecture

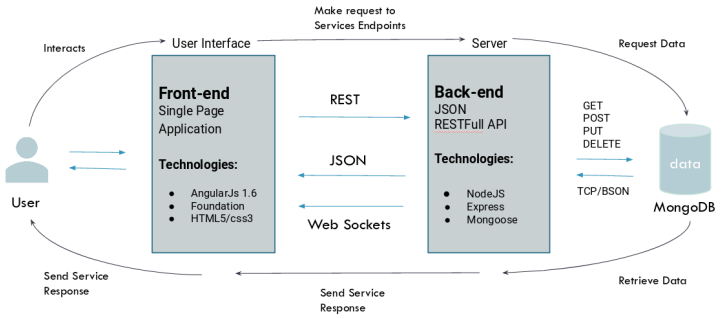


Figure 4.6: Uspector MEAN stack overall architecture

AngularJs, CSS Framework

On the client side, Uspector has a JavaScript layer that can freely communicate with web services on the server and, using the data from web services, make real-time updates to itself. For that purpose, JavaScript MVC framework AngularJs [1] is used. That decision leads us to design and build Uspector following patterns of Single-page web app architecture (figure 4.5b).

In order to get Uspector from prototype to production the CSS framework Foundation for Sites (version 6.5.3) was used. The choice of this framework enable us to make Uspector a responsive (i.e. various devices with different capabilities are supported), accessible, consistent (by following design patterns) and customisable platform.

Nodejs, Express, MongoDB, Mongoose

Uspector's back-end based on Node.js and MongoDB for the construction of server and database respectively. Both decisions led Uspector to be a scalable, flexible and fast platform.

Node.js [39] is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript for server-side scripting—running scripts to produce dynamic web page content before the page is sent to the user's web browser. It is also responsible to send requests and receive responses, as well as to interact with databases and files.

Express [4], is a web application framework for Node.js, released as a free and open-source software under the MIT License [25]. It is designed for building web applications and APIs. It also provides routing, view rendering and more. This framework provides I/O request handling and easy integration of third-party services and middle-ware, features which led to a faster development of the RESTFul API and more extensible server.

MongoDB [12] is a free and an open-source, cross-platform document-oriented database program. It is a NoSQL document database, meaning that MongoDB uses JSON documents with schemas. Also, since MongoDB supports data transfer through JSON format, data transfer from the web application is easy and economical. Moreover, JSON also allows easy client-server data transmission.

Mongoose [11] is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

Chapter 5

Uspector platform

This chapter outlines the details of the various front-end components that have been implemented in the context of Uspector, namely: (i) **User Authentication** (homepage) (ii) **User Workspace**, (iii) **Screen Viewer**, (iv) **Test Evaluation Session**. All visual elements and functionalities, of every system's component that the user interacts with, is described in the following chapter. Additionally, snapshots of the underlying functionality will be provided to illustrate the interaction of the user with the system.

5.1 User Authentication

Uspector is designed to be a web platform, accessible from any modern browser. By accessing system's public address (URL) users are landing in the *Homepage* (figure 5.1) of the system. The aim of this view is to introduce users in the platform's goal and drive them into the system's authentication process, through the login and registration forms.

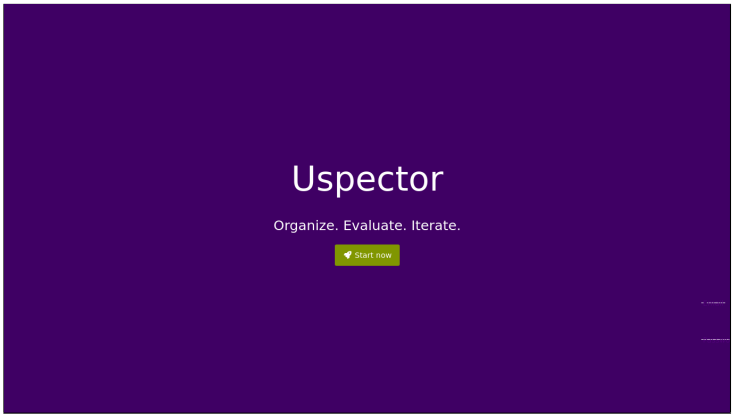
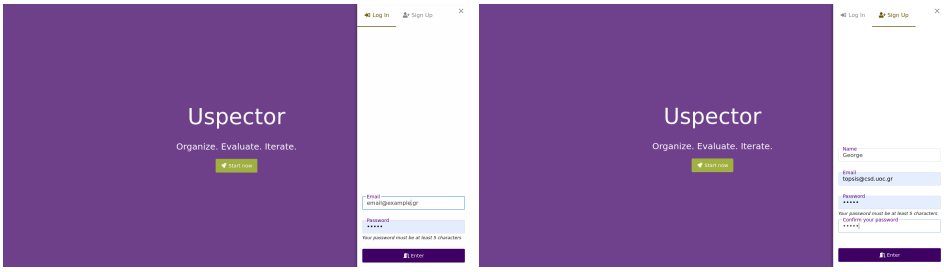


Figure 5.1: User Authentication in Homepage

Home View

User Authentication component enables on the one hand registered users to access their personal content (see login form at figure 5.2a) and on the other hand visitors to create an account to the Uspector platform (see registration form at figure 5.2b). Both functionalities comply with the functional requirements of system's features 1-3 on section 3.2.1.



(a) Login form (b) Registration form

Figure 5.2: Authentication forms in Homepage

5.2 User Workspace

Upon successful authentication, users enter into the restricted area of Uspector, which is the composite component of *User Workspace*. Its nature is not to represent a specific page of the system, but rather a ¡¡container¿¿ which define the main layout of app's UI and hosts the top bar (with account details and main navigation) and the active, according to current user's navigation, view (i.e projects, screens, usability tests etc). Through *User Workspace* component users acquire access and manage their and/or other's users content based on their role (owner or participant) on them. Moreover, users can navigate in the usability tests on which they participate as evaluators. Finally, users are able to set up their profile and configure the settings of the system.

Composite views consists of multiple different sub-views and as such ,the visual elements of *User Workspace* view are globally accessible components. As shown in figure 5.10 these visual elements are: (i) Uspector logo (also link to *User Dashboard*), (ii) main navigation menu, (iii) user profile and notifications information and (iv) toggle buttons for notifications and profile panels respectively.

5.2.1 User Dashboard View

User Dashboard view, as shown in figure 5.4, assists users to inspect the current state of their projects, designs and usability tests at a glance. Moreover, they can inspect their progress of usability tests in which participate as evaluators as well as a history review of their activity on the platform.

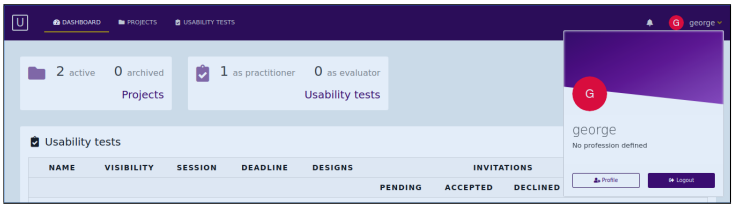


Figure 5.3: User Workspace - top bar

In particular, a practitioner of a usability test can view the current state of evaluation session (availability to evaluators, phase, deadlines), of invitations to other users (total numbers of each different state) and the project for which a usability test is conducted. Similarly, a user is able to view, for each one of his/her projects, its current state (active, archived), progress of its screens (states *In-Progress*, *Under evaluation* and *Approved*) as well as the number of usability tests that have/had been conducted on for it.

This view fulfils the functional requirements of features in sections 3.2.1.4 and 3.2.1.9.

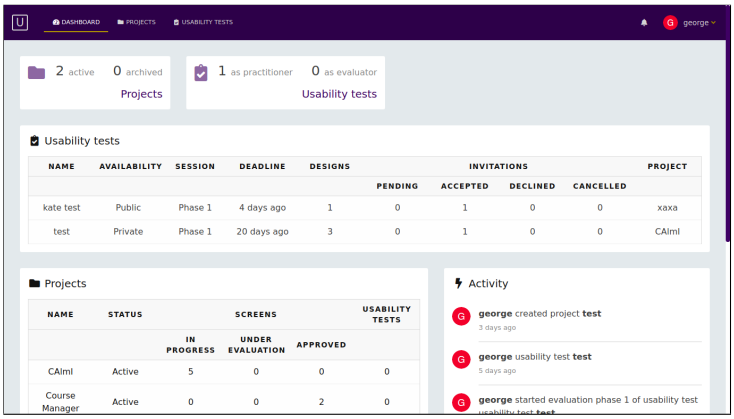


Figure 5.4: User dashboard view

5.2.2 Projects View

One of the platform's core views is the *Projects* view in which a designers can view and manage their collection of projects. As shown in figure 5.5 there are the following main visual components:

- Search box with floating label: As user types the query projects should be filtered by their name.

- Filter Forms: Filter projects by state (active, archived, favourite).

- Project creation button: A Cal-to-Action button which open a modal window to the user. This window contains a form with input elements for project's name and description (max 200 characters).

- Grid of projects: Responsive grid with projects as rectangular tiles. For each project, a brief summary about its details, to assist its owner to recognise it during his/her navigation, is provided. In particular, these details are to its name, its cover screen, total number of screens and the number of usability tests which are conducted for this project. Moreover, for each project a number of actions, such as (un)archive, (un)mark as favourite and deletion, are provided to its owner.

Projects view match the functional requirements about the display and management, both as individual entities and collection, of projects (features on section 3.2.2).

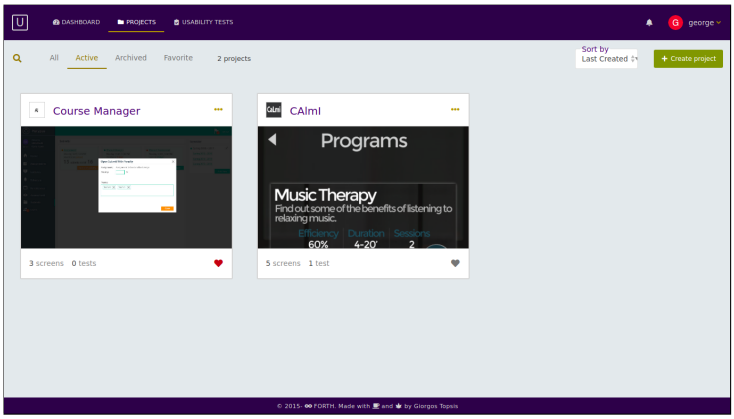


Figure 5.5: Projects view

5.2.3 Project Settings View

Designers are able set and update the details of a project such as project description and product's icon. figure 5.6 depicts the *Project Settings* view as this designed and implemented in order to comply with the functional requirements of feature 3.2.2.6.

5.2.4 Screens View

Each project consists of a set of screens which are presented in *Screens* view. Designers can create, update and delete screens according to their preferences. All functional requirements about the display and management, both as individual entities and collection, of screens (features of section 3.2.3) are satisfied by the following components of the UI (figure 5.7):

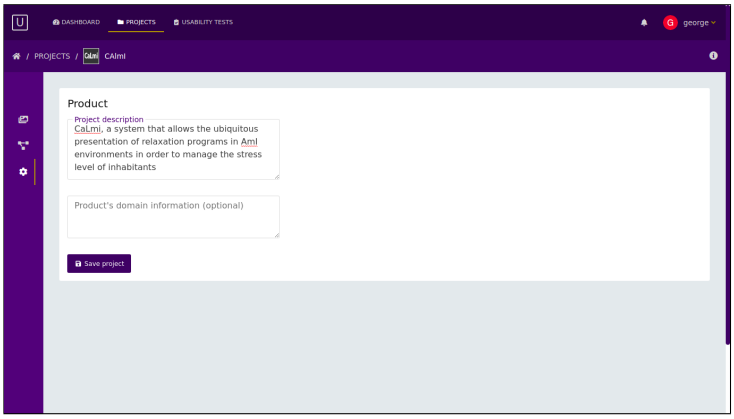


Figure 5.6: Project settings view

- Search box with floating label: As user types the query, screens should be filtered by their name.
- Filter Forms: Filter projects by state (in progress, under evaluation, approved)
- Tile view button: Open a panel for the configuration of the tiles' layout, such as the adjustment of dimensions' ratio and sizes.
- Screen creation button: A call to action button which allow designers to select design assets from their local storage in order to create multiple screens
- Grid of screens: Responsive grid with screens as rectangular tiles. Designers can also drag their designs assets from local storage to and drop them to this area in order to create multiple screens.

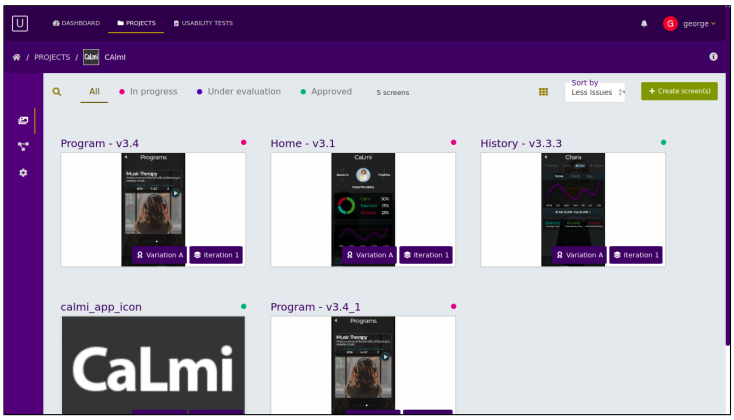


Figure 5.7: Screens View

5.2.5 Workflow View

While reviewing the project, there are various screens which look ready and good to implemented (development phase of a project), while there are others which they need a bit of revision before it will be approved and then, screens which their designers want to expose to evaluators through one or more usability tests. To segregate all of these screens in their respective status the user can navigate the *Workflow* view (figure 5.8) in which the screens can be marked as: (i) In Progress (Pink), (ii) Under Evaluation (Purple) and (iii) Approved (Green).

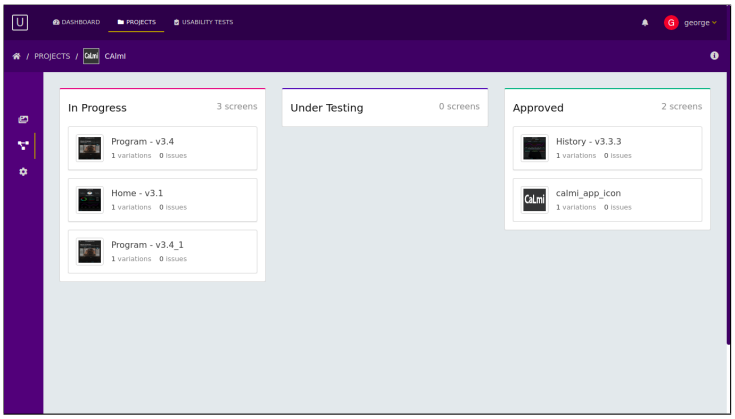


Figure 5.8: Workflow view

5.3 Screen Viewer

After loading a screen, the system displays the view *Screen Viewer*. Designers have the ability to manage screen's variations and versions as well as the usability issues which are associated with them. The main components of the layout are the following:

- Breadcrumb with expandable panel with the set of project's screens: Shows the path to the active screen and give the ability to designers to search, browse the remaining set of screens. The main goal of this panel is to let users to load an another screen in the viewer with a fast and effective way.
- View modes: Designers are able to load the viewer in different modes (i.e. inspection, hot-spots and comparison mode).
- Action buttons: Settings of viewer and export options
- Secondary navigation menu: Allow designers to navigate and manage (create, rename, delete) the variations and versions of the active screen.
- Viewer: The main component of the view which displays the active design (mockup of a specific version of a screen's variation).

- Inspection mode: View all reported issues from all usability tests and evaluators. For each reported issue, a practitioner is able to view a title, a brief description of the usability problem, the set of violated heuristics (only titles), a mean severity rating calculated by individual ratings (available only after *Phase 2*) as well as a suggested solution for the problem to designer, if that is provided by the issue author (evaluator). Moreover, a designer should be able to filter them based on the usability test, creator and its status (resolved or not).

- Comparison mode: Designers should be able to visually compare two different versions of the same screen's variation.

The importance of this view is obvious because of the large number of system's features which *Screen Viewer* implements. Functional requirements about the display and management, both as individual entities and collections, of variations and versions (see requirements of features are fulfilled.

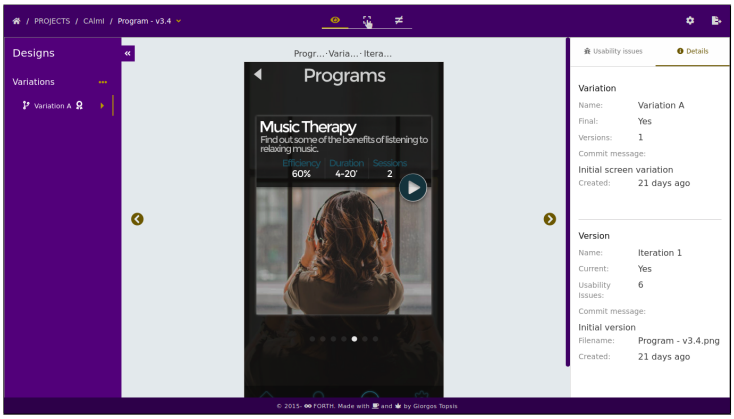


Figure 5.9: Screen viewer

5.4 Usability Tests View

The main layout of *Usability Tests* view is divided in two panels, the left one which contains and allow a practitioner to manage the set of usability tests which he/she conduct(ed) as well as the right one, which contains and similarly allow the practitioner to manage the set of heuristics collections (custom or not) which are available to tests.

Regarding the left panel, these components satisfy the functional requirements about the display and management, both as individual entities and collections, of usability tests (features on section 3.2.3). Similarly, in the right panel the functional requirements about the creation and management of heuristics collections are fulfilled (features on section 3.2.6.1).

Both panels address a user who behave as practitioner, who aims to create and conduct usability tests on projects of designs. More details about the visual elements and the functionalities of this view are presented below.

- Search box with floating label: As user types the query, usability tests should be filtered by their name.

- Filter Forms: Filter usability tests by user's role on them (practitioner, evaluator)

- Grid of usability tests: Responsive grid with usability tests as rectangular tiles. For each test the practitioner is able to view its name, its practitioner's profile photo, its current availability to evaluators (locked or not), the current state of its evaluation session and finally, a small list (up to five) of active (accepted the invitation) evaluators' personal photos and the project for which this test is conducted. Moreover, system allow practitioner to rename and delete a usability test.

- Button for Usability Test creation: A Call-to-Action button that presents the creation form modal window.

The right panel consists of the components regarding the presentation and management of heuristics collections. Therefore, the following components fulfil the functional requirements of system's features on section 3.2.6.

- List of heuristics collections: This list composed by both custom and custom (defined by the test's practitioner) and ready-made heuristics collections (J. Nielsen heuristics collection).

- Button for Heuristics Collection creation: A Call-to-Action button which make available a form in a modal window, which ask from the user to define the name of new heuristic collection.

5.4.1 Heuristics Collections Edit View

Practitioners are able to create, update and delete their custom heuristics collections as well as the content of them. A heuristics collection contains a set of heuristics guidelines/rules, where each one consists of a title and a brief description of this guideline. The goal of *Heuristics Collections Edit* view is to comply with the functional requirements of system's feature on section 3.2.6.1. It contains a form with collection name as well as a list of existing heuristics for a specific heuristics collection (figure 5.11).

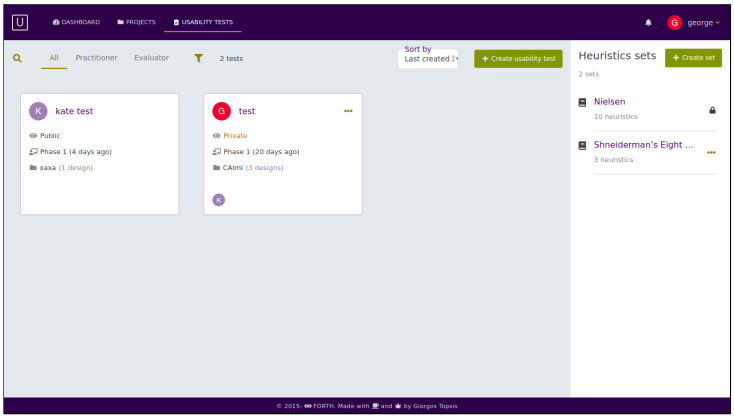


Figure 5.10: Usability tests View

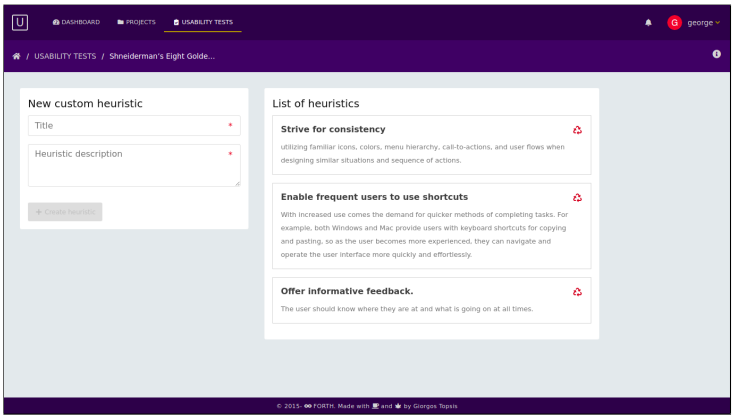


Figure 5.11: Heuristics collections edit View

5.5 Usability Test Conduction

The first visual element on the views relative to the conduction of a usability test is shown on figure 5.12. It presents the toolbar of a practitioner in order to control the availability and the phase of the evaluation session at any time..

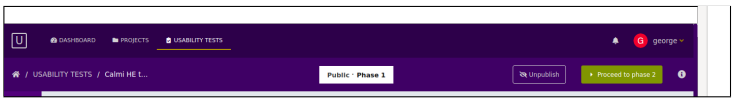


Figure 5.12: Toolbar for practitioner to control the progress of a usability test

5.5.1 Usability Test Dashboard View

During and after the conduction of a usability test system displays the statistics and the analytics of its progress in the *Usability Test Dashboard* view. A practitioner of a test has the ability to overview, at any time, the status of the evaluation session, the invitations (e.g. answers from invited users) as well as its availability to evaluators. As shown on figure 5.13, this view fulfils the functional requirements of feature 3.2.7.5.

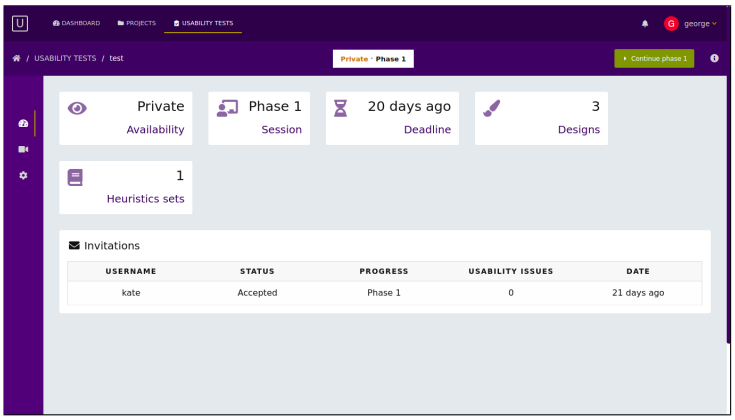


Figure 5.13: Usability test dashboard view

5.5.2 Usability Test Monitoring View

Apart from *Usability Test Monitoring* view a practitioner can monitor the progress of the evaluation through the *Usability Test Monitoring* view. More specifically, the organiser of the test is able to view which usability issues have been reported during and after *Phase 1* as well as he/she is able to inspect which one of them have been rated about the severity by all evaluators.

Moreover, after the completion of *Phase 1*, a phase in which evaluators report usability issues on the designs, the practitioner is able to read, edit, delete, merge and finally approve or disapprove a reported usability issue. Only approved issues will pass to *Phase 2* of the evaluation. As shown on figure 5.14 the above functionalities are fulfilled. *Usability Test Monitoring* view comply with the functional requirements of features on section 3.2.7.1.

5.5.3 Usability Test Settings View

Usability Test Settings view contains all configurations that a practitioner is able to do before starts the conduction of a test. More specifically, as defined on feature 10a these configurations are classified in four independent categories which are: (i) General Settings, (ii) Heuristics Settings, (iii) Designs Settings and (iv) Invitations Settings. Each one of them consists of required and optional settings.

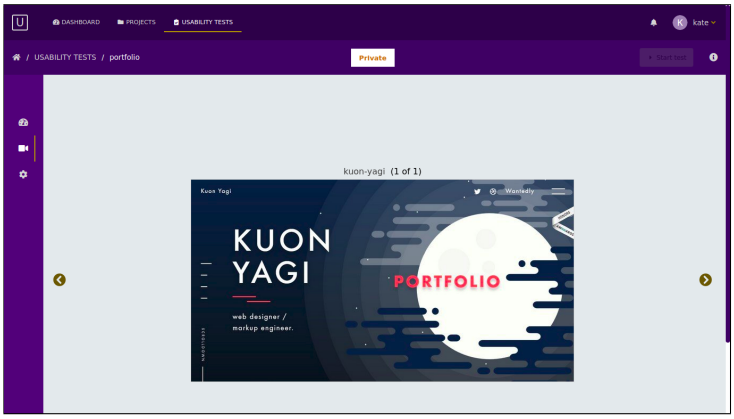


Figure 5.14: Usability test monitoring view

Below, details and descriptions about each category of usability test configuration are presented.

General Settings

The category of general settings refers to the list of configurations about the deadlines of each evaluation phase, the severity rating scale, as well as the optional definition of two scenarios of use. According to figure 5.15 the panel contains a form with the above components which fulfils the functional requirements of feature 3.2.7.1(a).

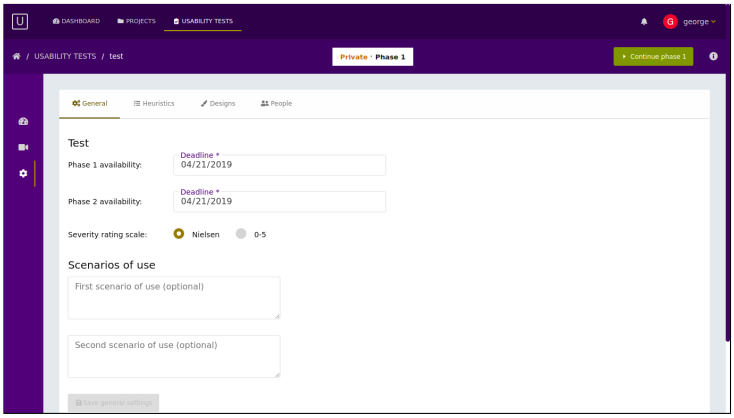


Figure 5.15: General settings form of a usability test

Selection of Heuristics

The second category of settings refers to the configuration of test's heuristics collections. A practitioner has the ability to use any heuristics collection which belongs

to him/her. Moreover, a practitioner can view the details (title, description) of every heuristic of a collection in order to make a decision to use it or not in a usability test. The main components of the view are (i) search box, (ii) used and available heuristics collections and action buttons to move a set of selected items from one set to the other (figure 5.16).

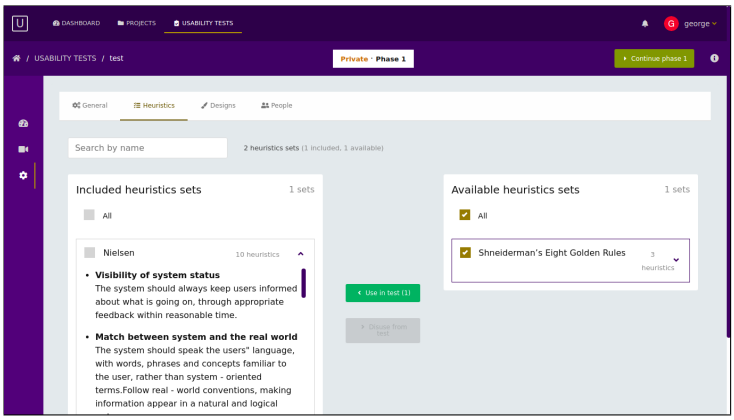


Figure 5.16: Heuristics management of a usability test

Selection of Designs

Similar to *Heuristics management* section the layout as well as the visual components are the same. The difference in this section is the content that a practitioner has to manage. Instead of heuristics collections, practitioner should include at least one design for evaluation in the usability test. As shown in figure 5.17 each list item consists of a design thumbnail, name of screen and its selected combination of variation and version.

In case a practitioner wants to change the selected design of a specific screen he/she should click the action button of the relative screen's item and interact with a form in a modal window. The main components of this view (figure 5.18) are:

- Details of current design, such as names for screen, final variation and current version. Also, a thumbnail of the design is provided as a visual hint to assist the practitioner to include it or not in the evaluation session of the test.

- Form to select a different variation and/or version of a specific screen. During this selection, there is a preview of the current design (combination of screen, variation and version) a practitioner is currently inspect.

Both views fulfil the functional requirements about the inclusion of one or more designs in a usability test during its configuration (features on section 3.2.7.1(b)).

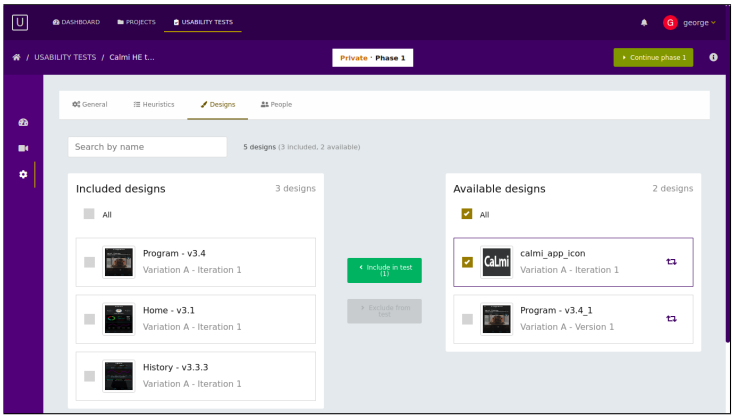


Figure 5.17: Design selection on a usability test

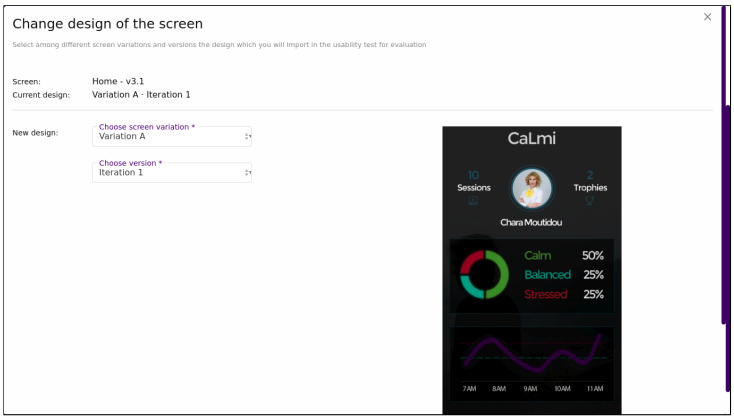


Figure 5.18: Selection of a different screen variant and or a different version of a specific screen

Invitations Management

The fourth category of settings refers to the participants of a usability test. To comply with the functional requirements of system's feature on section 3.2.7.1(c) the following views are implemented (figure 5.19 and figure 5.20).

- Search box: Search invitations by invited user's name as a practitioner types a query
- A grid of tiles about test's current invitations. This set consist of pending, accepted, cancelled and rejected invitations to/from users. For each invitation details about the invited user (name, personal photo, email), sent date, current state are provided. Moreover, a practitioner is able to cancel (after action confirmation) and share (via a private URL) an invitation.
- Button for user invitation: A Call-to-Action button which opens a modal

window and focus practitioner's attention on searching (by username or email) and sending invitations to users in order to participate in a usability test.

The task of user invitation for the participation in a usability test is critical to its successful conduction and completion.

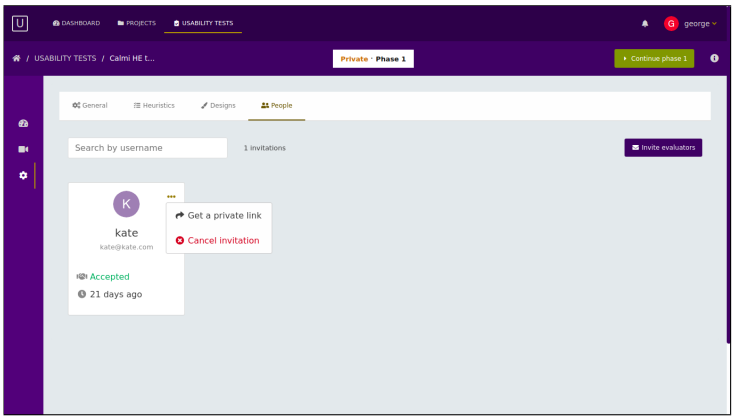


Figure 5.19: View with the list of current invitations

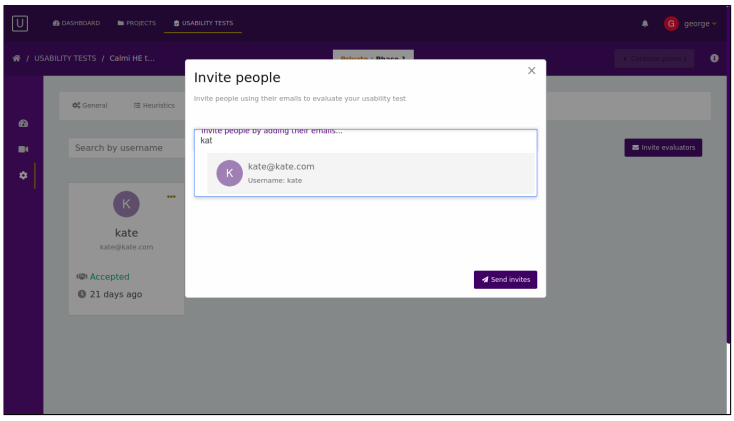


Figure 5.20: Search and invite users to a usability test

5.6 Test Evaluation Session

Test Evaluation Session view is accessible both by registered and from unregistered users who are invited to participate on a usability test. In either case, a user should be invited to participate on a usability session in order to deal with this view of the platform. Its main goal is to assist evaluators to find, report and rate usability issues on designs of the test. The main components of this view as shown in figure 5.21 are the following:

- Breadcrumb: Show evaluator the path to the usability test which she/he participates to.

- Evaluation progress badges: Messages which indicate the current availability of test and phase of the evaluation session.

- Toggle button for a panel with a form for evaluation session's configuration: an evaluator can configure the viewer by defining its color theme (light, dark, grey).

- Toggle button for a panel with the list of details about the available heuristics collections. These details relate to title and description of each heuristic.

- Manage progress button: Evaluators are able to define the completion of a evaluation phase.

- Viewer which contains the active design (mockup). Its dimensions depends on the designs digital ratio, width and height (in pixels). Viewer respect the image ratio and shrink it accordingly if at least one of image's dimensions overcome screens limits. Moreover, design viewer hosts the set of reported usability issues for this digital asset.

- Usability issue popover: Popovers contain information about usability issues according the current phase of evaluation session. The whole set of issue's details composed by its title, set of violated usability heuristics, a brief description of the issue and the severity rating of the its author (only after *Phase 2*).

- Navigation buttons: Evaluators can browse among designs of a test during the evaluation session.

The above components of these views fulfil the functional requirements of system's features on section 3.2.7.1.

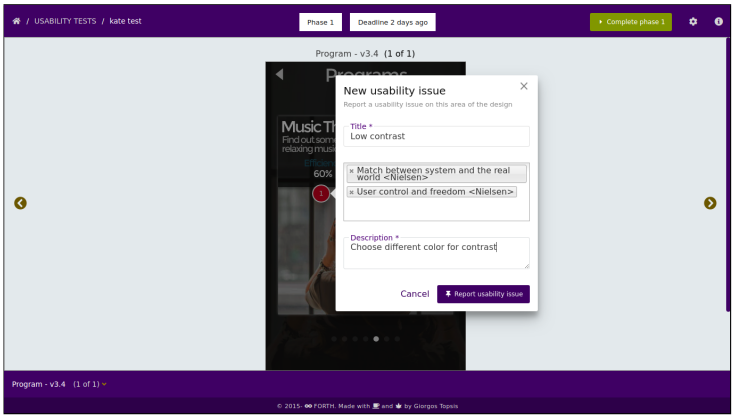


Figure 5.21: Evaluation session - evaluator reports a usability issue

5.7 Utilities

5.7.1 User Profile View

Profile view contains all the personal information of a user as well as his/her preferences on system settings. As shown in figure 5.22 each tab contains configuration option of a different area of settings. More specifically, users can update the their profile pictures, profession details, links to personal social media profiles/pages, and their entry password. Moreover, they are able enable or disable email notifications from the system. This view matches the functional requirements of system's features 6 and 7 on 3.2.1.

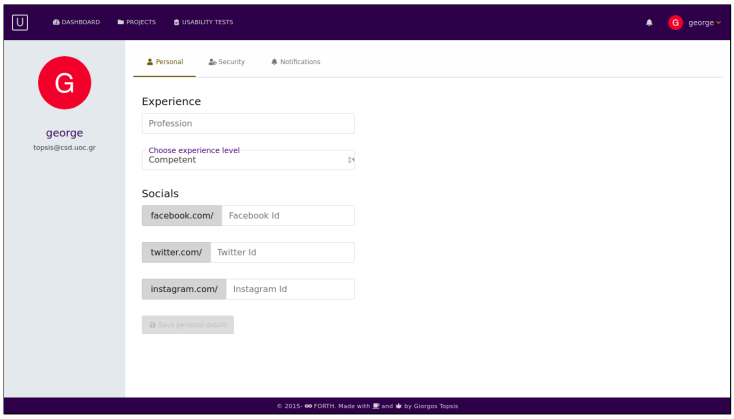
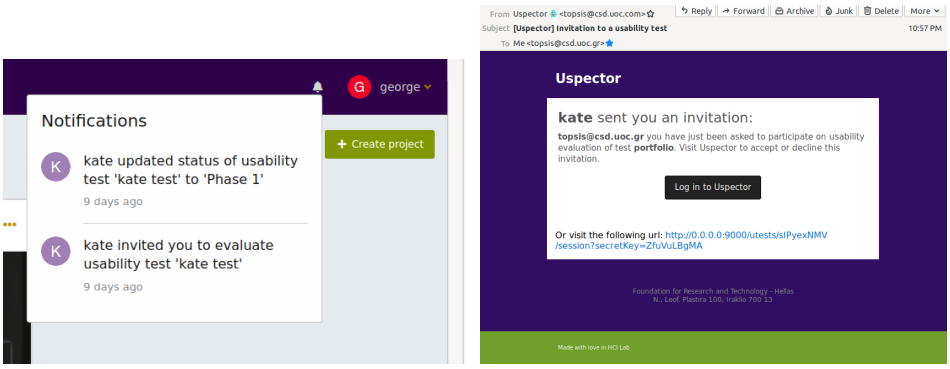


Figure 5.22: User profile view

5.7.2 Push and Email Notifications

System notifications are an important feature of the platform. Users are notified and are interrupted from their current task only when an important event occurs,

like an invitation for the participation on a usability test. The notifications panel on the main top bar displays all (read and unread) notifications of a user (figure 5.23a). Moreover, a user will receive a personalised email (figure 5.23b), in order to participate on a usability test. The following figures show the outcome of system's notification mechanism which has been designed and implemented in order to conform to the functional requirements of system's features on sections 3.2.1.5 and 3.2.1.8.



(a) Notifications on invited users (b) Email to Invited user

Figure 5.23: Push and email notifications

Chapter 6

Evaluation

The objective of this chapter is to report the set-up, the experiment and outcomes of an evaluation that attempted to assess the Uspector platform. The main goal of this process is to evaluate if the system assist designers to organise and evaluate their designs. We followed the three key steps of Usability Testing as shown in figure 6.1.

This section will present: (i) the evaluation methodology, profile of evaluators and tools which we used (step 1) (ii) the process of the experiment (step 2) and (iii) the findings about the usability of Uspector platform in terms of the number and the severity of the reported usability issues (step 3).

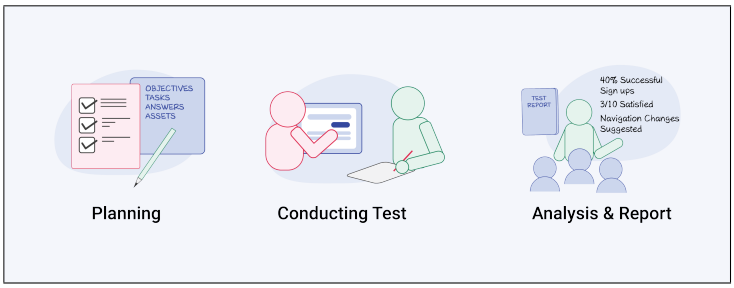


Figure 6.1: Usability Ttesting steps of Uspector evaluation experiment

6.1 Evaluation Methodology

Before proceeding to a formal user-based evaluation of the platform we decided to conduct a usability test using the Cognitive Walkthrough (CW) methodology [32] in order to reveal possible major usability issues of our digital product. We followed the steps as presented in the website of Interaction Design Foundation [20]. Our main goal was to see whether or not a new user can easily carry out tasks within our system. Therefore, we were looking for an extremely cost-effective, compared to many other forms of usability testing, fast and task-oriented, in contrast to

HE which is a more holistic usability inspection, methodology to get the most severe usability issues. One more aspect which assist to made our decision was the fact that using the CW methodology we would drive evaluators to focus on finding usability issues which are related to end-users' most common tasks instead of spending time of the evaluation trying to inspect all the system's functionalities.

6.2 The experiment

6.2.1 Participants

Our team decided to recruit only two participants for the first usability test in Uspector due to time limitations. A small but satisfactory number of participants in order to reveal a significant portion of possible usability problems according to Nielsen [45]. The task was to find participants with a specific user profile (expertise, age, experience etc). Finally, evaluators consist of both genders (one female and one male) and both of them belong to the age group of 30-40 years old. Moreover, they have a high expertise on the domain of HCI. All participants are familiar with the iterative design process and the principles of HCI as well as the domains of version control systems and usability testing methodologies and platforms.

The two participants' responsibilities were to attempt to complete a set of representative task scenarios presented to them in as efficient and timely a manner as possible, and to provide feedback regarding the usability and acceptability of the user interface. The participants were directed to provide honest opinions regarding the usability of the application, and participated participate in post-session debriefing sessions.

6.2.2 Use Case Scenario and Tasks

A critical task about our try to evaluate the Uspector platform was to define a flexible, short, simple and understandable set of "connected" tasks which construct a realistic scenario of use Uspector. This set should have represented the most common user flows in order to get the most valuable findings about the way a regular user would use our system, what usability issues(s) he would deal with and what is the overall user experience.

As a result, we ended up with a set of forty four (44) tasks, with each one having a descriptive title and a brief description. The final use case scenario browse a large portion of the system and it is presented in Appendix A. For each one of the above set of tasks, the following four questions [32] have been asked for each evaluator.

- Question 1 (Q1): Will the user try and achieve the right outcome?
- Question 2 (Q2): Will the user notice that the correct action is available to them?

- Question 3 (Q3): Will the user associate the correct action with the outcome they expect to achieve?
- Question 4 (Q4): If the correct action is performed; will the user see that progress is being made towards their intended outcome?

6.2.3 The process - Evaluation Sessions

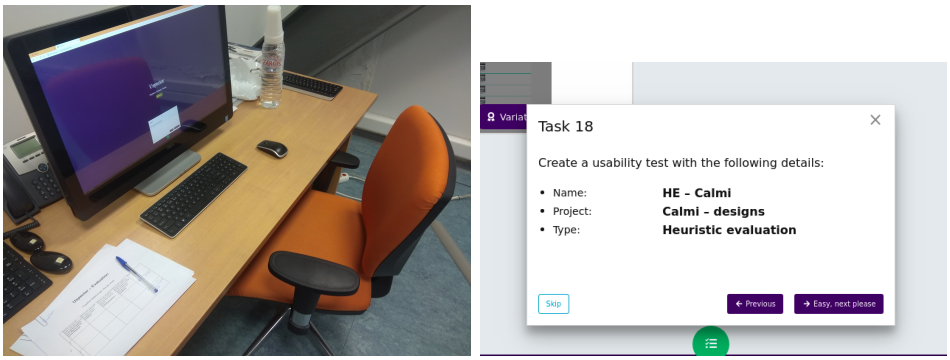
The usability test that was conducted, had four stages: (i) the preparation, (ii) the introduction, (iii) the actual test and (iv) the debriefing session.

Preparation

During the preparation stage we set-up the desktop computer to the initial state of Uspector. We decided not to use external artefacts or recording devices, such as cameras, microphones or one-way mirrors. As regards the practitioner of the experiment, he only needed a pen and a paper sheet with the lists of tasks that the evaluator should carry out during the session.

As figure 6.2a shows, the main set-up of the room consisted of a desk, a chair, a desktop PC (display, keyboard and mouse) with a modern browser and an internet connection. More specifically, we used a desktop PC with the operating system Windows 10 [28] and as browser we used Chrome [23].

The participants received an overview of the usability test procedure, equipment and software. Aiming to further assist the assessors with the process of evaluation, the tasks were not written in paper sheets, as the common practice commands, but instead they were provided to them inside Uspector itself using an integrated popup tool as figure 6.2b shows. Participants were able to read and navigate among tasks during sessions without time limits. There were also a backup plan in case of an error with the relative feature of our system, which was the use of an electronic presentation (Microsoft Office presentation).



(a) Set up of the evaluation session (b) Integrated tool for tasks presentation

Figure 6.2: Evaluation session

Introduction

Before the conduction of the experiment and after the preparation stage the evaluators were invited in the classroom, one by one in separate sessions. We introduced ourselves and discussed about the process,their roles and the final goal of the evaluation process. Subsequently, we explained to them the system's goal as well as product's domain.

The Test

During the test the practitioner was observing and listening user's actions and suggestions carefully so that he identify user's emotions. Finally, he was trying to answer the four questions for each task that the evaluator should have carry out. Moreover, he was keeping notes about possible usability issues which may or may not have a direct relation with a specific task of the scenario. This method led to a significant number of questions which the practitioner had the opportunity to ask the evaluators in the debriefing session.

The Debriefing Session

The last stage was the most informal due practitioner and evaluators had a conversation about their experience using the system. Part of the below findings emerged from this session which indicates the importance of this stage in a usability test.

6.3 Evaluation Results and Findings

During the conduction of the test the participants were operating the scenario's tasks and at the same time the practitioner was recording not only the successful completion of each task but also the details (about the reason) behind an unsuccessful task's completion. Each assumption about the reason of a failure based also on participants feedback and thoughts as part of their experience with our platform.

On table 6.1 we present details (the question (Q1-Q4) that a participant's actions did not "answered" correctly) about the set of tasks (scenario of use on Appendix A) which did not fully completed by at least one of the participants. Moreover, we state which one of them have been resolved after a revision that has been applied after the conduction of the usability assessment.

Table 6.1: Incomplete tasks of test's scenario from at least one participant

Task ID	**Task Description**	**Participant 1**	**Participant 2**	**Resolved**
1	Log in to the system	Q2 - Expected login form submission on keystroke Enter	Q2 - Big distance between form and trigger button	Yes
2	Change and set the profile picture	Yes	cannot find upload button	Yes
5	Create a new project	Q2 - Cannot find action's trigger button	Q2 - Cannot find action's trigger button	Yes
6	Change application icon of the project	Q1 - Cannot understand what a screen is	Q2 - Cannot find action's trigger button	Yes
8	Select and delete screens	Q2 - Cannot find action's button	Yes	Yes
12	Load screen TV and create a screen variation using file	Q1 - Confusion between screen and variation	Q2 - Cannot find trigger button	No
13	View details for a design	Q1 - Confusion between screen and variation	Cannot find trigger button	No
15	Mark project Calmi designs as Favourite	Q2 - Cannot find action button "Favorite"	Yes	Yes
16	Yes	Q4 - Cannot view system's feedback	Yes	Yes
21	Create a custom set of heuristics	Q2 - Cannot find trigger action button	Yes	No
23	Search and invite a user to a test	Q3 - Cannot understand the label/signifier	Yes	Yes

Yes				
30	Report a usability issue	Q2 - Cannot find the way to add an issue upon a design	Yes	No
42	Set a screen as a project's cover	Q1 - Cannot understand what a cover means	Q2 - Cannot find action button "Set screen as cover"	Yes

One of the most severe usability issue this experiment revealed is related to the architecture of the information of Uspector platform. In particular, there are some navigation problems based on the relations of entities *Screen*, *Variation* and *Version*. Participants could not find how these pieces of information are interconnected and how they are going to manage them through our platform. Several times during the evaluation session, there was not convergence between their mental and system's models about these entities. As a result, this overall confusion led to mistakes during their navigation in the app, in cases they were managing screens although their intentions were to manage variations and/or versions and and vice versa.

Similarly, one more navigation problem that was uncovered, is that participants have difficulties to find their heuristics collections. Their expectations was to find and manage all (custom) heuristics during usability tests' configuration and not before. That issue refers to the relation between entities *Usability test* and *Heuristic Collection* as those have been defined in chapter 4 about the data modelling of Uspector platform.

Furthermore, the execution of the scenarios' tasks by the participants uncovered issues about the readability and discoverability of signifiers, icons and buttons labels, mainly due to their small size of font.

Finally, the following notes were gathered by the us, as test organizers/moderators, both during the evaluation session as a result of participants' actions on several tasks and during the debriefing sections after the end of the each experiment. In particular, one of the participants expected to be able to define the order in which the designs of a test will be available to the evaluators. Also, both of them encountered difficulties on tracking the progress of a test when they participate on a usability test as evaluators.

In general, it was clear to participants the overall goal of Uspector platform as well as how this system aims to assist designers during the iterative design processes for the development of a service/product.

Chapter 7

Summary and Future Work Directions

In this chapter we summarise the proposal of Uspector platform and present our future directions with respect to its features and interaction paradigm.

7.1 Summary

Over the last few years, getting bigger and bigger the portion of the industry that have recognised the role of customers' overall user experience; before, during and after their interaction with a service/product, for the success of a company's goal. At the same time, in order a company to remain competitive in the market, it needs to apply fast-paced production processes so that it releases new features or brand new services/products. Otherwise, customers will choose a similar service/product from a competitor. Under those circumstances, design teams of companies and organisations look for solutions to deal with the continuous management and usability testing of the designed mockups.

To that end, this thesis presents Uspector, a web platform that aims to accommodate the needs of designers, practitioners and evaluators during the iterative user-interface design process of a product. In particular, through Uspector: (i) designers can organise their creative content in projects of screens, with every screen consisting of a set of variations, and they can keep track of their evolution over time; (ii) practitioners can efficiently organise and conduct usability experiments using the Heuristic Evaluation methodology in order to uncover possible usability issues of designs; and (iii) evaluators can be supported while inspecting designs for errors.

From an engineering perspective the Uspector platform: (i) introduces a centralised storage for all design files, plus an easier way for designers to organise (projects, define variations), search and sync their mockups, (ii) provide a version control system so that designers track the changes of their work, (iii) offers a dashboard to practitioners to organise, configure, conduct and analyse a usability test

and (iv) provide an editor to evaluators in order to report and, subsequently, rate the severity of reported usability issues in a design.

Uspector has been tested in a case study of iterative design. In particular, we have conducted a usability experiment using the methodology of Cognitive Walkthrough. With the assistance of two HCI experts we identify a significant number of usability issues regarding the visibility of several signifiers/ and buttons' labels and the readability of important visual elements (typography issues) of our application User Interface. An important insight was the major confusion of evaluators about relations among *Screen*, *Variation* and *Version* which is a high priority for further investigation and possible refinement. In general, the case study has confirmed that Uspector platform could eventually assist designers to centralise and organise (projects, screens, designs variations) their creative content, uncover possible usability issues by conducting multiple low-cost and rapid usability tests, as well as inspect the evolution of their work through Uspector's version control system.

7.2 Future Work

The Uspector platform designed and implemented using effective and broadly-used software patterns and technologies. More specifically, using AngularJs, RESTFul API and following Component-Based development practices Uspector has been structured to be a modular application, a feature which assist other designers and developers to extend its functionality by adding new features. Moreover, its information architecture enables the addition of external building blocks which will not affect the clarity and intuitiveness of system's navigation.

Under those circumstances, there are some enhancements and directions which could be followed in the continuation of platform's development in order to empower practitioners in a more effective way, conduct advanced heuristic evaluation tests and support other evaluation methodologies (user testing) as well.

7.2.1 Empower Practitioners of Usability Tests

Practitioners of usability tests are already supported by automating and accelerating both the process of configuration and the process of conduction of a test. Nevertheless, there is a big part of work related to the aggregation, cleaning and filtering of the results after the phase 1 of a evaluation session which practitioners are responsible to carry out. For example, after the completion of phase 1 by all evaluators in a usability test, its practitioner should remove empty or invalid usability issues reports, resolve typography errors on them or even merge reports which refer to the same usability issue.

Having that in mind, an addition to the platform could be a semi-automated mechanism which calculate the similarity probability of two reports, from the same or different evaluators, to be a duplicate report. In case of a pair with a probability greater than a specific threshold then system notify the practitioner

and ask his/her confirmation to merge these two reports by keeping the data of each one at the same time. A proposed algorithm to support this feature could be to calculate the probability function mentioned above as the distance between two spots of the reports upon a design and the intersection between the heuristics set that each issue violates.

7.2.2 Heuristic Evaluation using Multiple Screen Variations

An equally interesting feature could be the support of an evaluation including two or more screen variations. So far, in Uspector there is the limitation to select a version from only one variation of a specific screen in order to include it in a usability test for evaluation. Therefore, designs from two different screen's variations cannot exist in the same usability test. This design decision currently constrains practitioners who want for example to examine which screen's variation would lead to better usability results in a specific user flow (series of screens) of their product.

System could overcome this limitation and allow practitioners to include designs from different screen's variation by inserting the notion of *user paths* on which one and only one screen's variation can be included. In that way, a usability test would consists of a number of *user paths*, a number which is function of the number of screens and the number of variation each one of them has included in a test. For example, if we have two (2) screens with one (1) and three (3) variations respectively then this test will consist of six (2 x 3) different *user paths*. Moreover, the practitioner of a test could split the set of evaluators in six (6) subsets and support the comparison of each path's results.

7.2.3 Prototypes - User Testing

A major direction and challenge would be to integrate and support and other usability methods, besides heuristic evaluation. More specifically, Uspector could support the creation of prototypes for each project and so the conduction of user testing on them by candidate end-users of a product.

Due to high modularity of Uspector, the design and development of this functionality in the existing code-base is not difficult. More specifically, *Screen Viewer* view could support one more tab panel to support *Hot-spots* mode in order to give the ability to designers to "connect" their screens by defining hot-spots area on them. As a result a connected set of designs would be a prototype of designer's project. Subsequently, a practitioner could conduct a usability test by either using heuristic evaluation or user testing methodology.

7.2.4 Pilot study of Uspector platform

A more extended and complete usability evaluation experiment should be conducted on Uspector tool. In particular, a second round of the evaluation test that we conducted have to be conducted with a bigger set of participants (up to five), in order to validate the changes that the results of the first round had triggered.

Moreover, a user testing with end users, who will be professional designers and usability testing coordinators, could be conducted in order to reveal extra possible usability problems in the overall user experience that Uspector platform offer to its users.

Bibliography

[1] AngularJS — Superheroic JavaScript MVW Framework. https://angularjs.org/. [Online; accessed 4-May-2019].

[2] Daprota M2 MongoDB Data Modeling Adviser.

[3] The digital design toolkit.

[4] Express - Node.js web application framework. https://expressjs.com/. [Online; accessed 4-May-2019].

[5] Figma: the collaborative interface design tool. https://www.figma.com/. [Online; accessed 4-May-2019].

[6] Folio - Simple visual version control tool for Mac based on Git. http://folioformac.com. [Online; accessed 6-May-2019].

[7] Git. https://git-scm.com/. [Online; accessed 9-May-2019].

[8] GitHub - angular-fullstack/generator-angular-fullstack: Yeoman generator for AngularJS with an Express server. https://github.com/angular-fullstack/generator-angular-fullstack. [Online; accessed 4-May-2019].

[9] InVision | Digital product design, workflow & collaboration. ttps://www.invisionapp.com/. [Online; accessed 6-May-2019].

[10] Kactus. https://kactus.io/. [Online; accessed 23-April-2019].

[11] Mongoose ODM v5.5.5. https://mongoosejs.com/. [Online; accessed 4-May-2019].

[12] The most popular database for modern apps. https://www.mongodb.com/. [Online; accessed 4-May-2019].

[13] Simple and powerful media asset management. https://pics.io/hello. [Online; accessed 4-May-2019].

[14] Sketch version control & design workflow management.

[15] Streamline your interface review process – Capian. https://capian.co/. [Online; accessed 8-May-2019].

[16] Trunk Version Management Software For Designers.

[17] UX Check. https://www.uxcheck.com. [Online; accessed 8-May-2019].

[18] UX Quiz. http://uruit.com/ux-quiz/. [Online; accessed 8-May-2019].

[19] Versions: Git for Designers.

[20] How to Conduct a Cognitive Walkthrough . https://www.interaction-design.org/literature/article/how-to-conduct-a-cognitive-walkthrough, 2019. [Online; accessed 20-October-2019].

[21] New file format in Sketch 43 . https://sketchplugins.com/d/87-new-file-format-in-sketch-43, 2019. [Online; accessed 20-October-2019].

[22] Balsamiq - a wireframing tool. https://balsamiq.com/, 2019. [Online; accessed 20-October-2019].

[23] Chrome browser. https://www.google.com/chrome/, 2019. [Online; accessed 20-October-2019].

[24] Go Visually. https://govisually.com/, 2019. [Online; accessed 20-October-2019].

[25] MIT - License. https://opensource.org/licenses/MIT, 2019. [Online; accessed 20-October-2019].

[26] Mockflow tool. https://mockflow.com/, 2019. [Online; accessed 20-October-2019].

[27] Plant – version control app and Sketch plugin for designers. https://plantapp.io/, 2019. [Online; accessed 6-May-2019].

[28] Windows 10. https://www.microsoft.com/el-gr/windows/, 2019. [Online; accessed 20-October-2019].

[29] Terence S. Andre, H. Rex Hartson, Steven M. Belz, and Faith A. Mccreary. The user action framework: a reliable foundation for usability engineering support tools. *International Journal of Human-Computer Studies*, 54(1):107–136, January 2001.

[30] Carmelo Ardito, Rosa Lanzilotti, Paolo Buono, and Antonio Piccinno. A tool to support usability inspection. volume 2006, pages 278–281, 01 2006.

[31] Robert W. Bailey, Robert W. Allan, and P. Raiello. Usability testing vs. heuristic evaluation: A head-to-head comparison. *Proceedings of the Human Factors Society Annual Meeting*, 36(4):409–413, 1992.

[32] Marilyn Blackmon, Peter Polson, Muneo Kitajima, and Clayton Lewis. Cognitive walkthrough for the Web. volume 4, pages 463–470, April 2002.

[33] A. S. Carter and C. D. Hundhausen. How is User Interface Prototyping Really Done in Practice? A Survey of User Interface Designers. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 207–211, September 2010.

[34] Chattratichart, Jarinee, Jacqueline Brodie, and Jacqueline. Extending the heuristic evaluation method through contextualisation. *Human Factors and Ergonomics Society Annual Meeting Proceedings*, 46:641–, 09 2002.

[35] Jarinee Chattratichart and Gitte Lindgaard. A Comparative Evaluation of Heuristic-based Usability Inspection Methods. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, pages 2213–2220, New York, NY, USA, 2008. ACM. event-place: Florence, Italy.

[36] +Piyas De. Single Page Application with Angular.js, Node.js and MongoDB (MongoJS Module).

[37] A. Dhouib, A. Trabelsi, C. Kolski, and M. Neji. A classification and comparison of usability evaluation methods for interactive adaptive systems. In *2016 9th International Conference on Human System Interactions (HSI)*, pages 246–251, July 2016.

[38] User focus. Userfocus - 20 search usability guidelines. `https://www.userfocus.co.uk/resources/searchchecklist.html`, 2019. [Online; accessed 20-October-2019].

[39] Node js Foundation. Node.js. `https://nodejs.org/en/`. [Online; accessed 4-May-2019].

[40] V. Grigoreanu, R. Fernandez, K. Inkpen, and G. Robertson. What designers want: Needs of interactive application designers. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 139–146, September 2009.

[41] Ling, Guanhua Chen, Salvendy, and Gavriel. Extension of heuristic evaluation method:a review and reappraisal. *Ergonomia - An International Journal of Ergonomics and Human Factors (IJE*, 27:179–, 01 2005.

[42] Rolf Molich and Jakob Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3):338–348, March 1990.

[43] Mark W Newman and James A Landay. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. page 12.

[44] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158. ACM, 1994.

[45] Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 206–213, New York, NY, USA, 1993. ACM.

[46] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 249–256, Seattle, Washington, United States, 1990. ACM Press.

[47] Stephen Oney, John Barton, Brad Myers, Tessa Lau, and Jeffrey Nichols. Playbook: Revision Control and Comparison for Interactive Mockups. In Maria Francesca Costabile, Yvonne Dittrich, Gerhard Fischer, and Antonio Piccinno, editors, *End-User Development*, volume 6654, pages 295–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[48] Fatih Kursat Ozenc, Miso Kim, John Zimmerman, Stephen Oney, and Brad Myers. How to support designers in getting hold of the immaterial material of software. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, page 2513, Atlanta, Georgia, USA, 2010. ACM Press.

[49] Helen Petrie and Lucy Buykx. Collaborative heuristic evaluation: improving the effectiveness of heuristic evaluation. 11 2019.

[50] usabiliTEST. usabiliTEST: Products. `https://www.usabilitest.com`. [Online; accessed 8-May-2019].

Appendix A

Evaluation process - Scenario of use

Table A.1: Set of tasks

Task id	Description
1	Log in to the system using the credentials: • username: practitioner@test.abc • password: practitioner
2	Change and set the profile picture using the following file: "profile-picture.png"
3	Turn off email notifications
4	Create a new project.
5	Give to this new project the followings details: • Name: Calmi • Description: Calmi is a master thesis at HCI lab
6	Change application icon of the project using the following file: "calmi_app_icon.png"
7	Create five (5) screens in the project using the following files (from folder v1): • "History - v1.2.1.png" • "Home–v1.1.png" • "Notifications - v1.png" • "Program - v1.5.png" • "Users-v1.png"

8 Select and delete the following screens:
- History - v1.2.1
- Notifications – v1

9 Rename the following screens:
- Home – v1.1 to Home page
- Program – v1.5 to Program page
- Users - v1 to Users page

10 Enlarge the titles of the screens, order screens reverse-alphabetically

11 Create a new screen using file "v1 / TV - A – v1.png" and the following details:
- Screen name: TV
- Variation name: Variation A

12 Load screen TV and create a screen variation using file "v1 / TV - B – v1.png" and the following details:
- Variation name: Variation B
- What differs this variation from others?: Dark theme

13 View details for the design: Program - variation A - iteration 1

14 Rename project Calmi to: Calmi designs

15 Mark project Calmi designs as Favorite

16 Create a custom set of heuristics with the name "Mobile Heuristics" and add the following heuristic:
- First heuristic
- Title: Prioritisation of Function Over Form
- Description: Design decisions are driven by what an element is meant to do rather than prioritising its visual style

17 Second heuristic
- Title: Availability of Information
- Description: The strategic placement of interface elements at users' fingertips so they don't have to rely on memory

18 Create a usability test with the following details:
- Name: HE – calmi
- Project: Calmi – designs
- Type: Heuristic evaluation

19 Configure and save general settings of test HE – Calmi with the following details:
- Phase 1 deadline: 20/04/2019
- Phase 2 deadline: 30/04/2019
- Severity rating scale: Nielsen

20 Use the following sets of heuristics in the test:
- Nielsen
- Mobile – Heuristics

21 Include the following designs in the test:
- Home page: variation A - iteration 1
- Program page: variation A - iteration 1

22 Change the selected design for the screen TV: Select Variation B, Iteration 1 and include it for evaluation in the usability test

23 Search and invite "evaluator@evaluator.abc" to evaluate test HE – Calmi Start test

24 You want to login to the system using the following credentials:
- Email: evaluator@evaluator.abc
- Password: evaluator

25 Read new notifications

26 Accept the invitation from user practitioner for participation in the usability evaluation of test HE – Calmi

27 Begin the evaluation of test HE - Calmi

28 Report the following usability issues:
- Screen: Program Page
- Title: Small negative space between two action buttons
- Heuristics: Error prevention
- Description: The negative space between action button "more" and select program button is too small which will drive to high error rates from user

29 Report the following usability issue:
- Screen: Home Page
- Title: Wrong font on title
- Heuristics: Consistency and standards
- Description: Font of page title should be consistent with the main font of the system

30 Report the following usability issue:
- Screen: TV Page
- Title: Not relevant icon - label on menu item
- Heuristics: Match between system and the real world
- Description: The combination of smile icon and label "Program" do not indicate the functionality of finding a collection of possible media which will relax user

31 Complete phase 1 of evaluation

32 View the progress of all evaluators in test HE – Calmi
Specifically, in which phase of evaluation each one is working on

33 Proceed to evaluation phase 2 of the usability test HE – Calmi

34 Rate each of the reported usability issues in all designs of the test with the following:
- Screen: Program Page
- Severity: Cosmetic problem only: need not be fixed unless extra time is available on project
- Suggested solution: Use the same font for all content across the pages of the screen

35 Screen: Home Page
- Severity: Major usability problem: important to fix, so should be given high priority
- Suggested solution: -

36 Screen: TV Page
- Severity: Minor usability problem: fixing this should be given low priority
- Suggested solution: -

37 Complete the evaluation phase 2 and then logout

38 Load screen Home page and view existing usability issues

39 Use the file "v2/Home – v2.1.png" to create a new iteration for variation
Variation A of screen Home Add the following details:
- What you were working at ?:
- Resolve existing issues about color contrast of graph

40 Mark variation Variation B of screen TV as the Final variation

41 Mark screen Home as ready

42 Set screen TV as cover of project Calmi designs

43 View the workflow of project Calmi designs

44 Archive project Calmi designs and then logout.