

École polytechnique de Louvain

Bootstrapped Evaluation: A Mobile Application for Evaluating the User Interface of Mobile Applications

Author: **Alberto SAMELE**
Supervisor: **Jean VANDERDONCKT**
Readers: **Nicolas BURNY, Mathieu ZEN**
Academic year 2021–2022
Master [60] in Computer Science

Abstract

Following the advent of desktop and laptop devices, a new type of tool found its way in the hands of a staggeringly high number of people: smartphones. By only being a pocket-reach away, it is not hard to see the appeal that this relatively new type of device poses for the general population. This, aided by constant improvements of network technologies, both in terms of coverage, speed, and reliability, led us to having, as of 2020, over 68% of web traffic being generated by mobile devices alone, and this number is only projected to increase.

With about a thousand new applications being pushed on the Apple's AppStore alone every day and with the average user spending only approximately 50 milliseconds evaluating the potential interest of a new app, the development of appealing, pleasurable user interfaces becomes fundamental to win over the hearts and minds (and downloads) of people, which often value beauty more than utility or quality.

Given these premises, it becomes necessary to have a systematic approach to evaluate the fitness of user interfaces, as what cannot be measured can't be improved either.

To keep the measuring process as quick, easy and immediate as possible, a mobile application capable of evaluating graphical user interfaces based off of several aesthetics metrics has been developed, allowing us to give a numeric rating to the beauty of GUIs with nothing more than a screenshot, capability which is particularly useful in case one were to rate user interfaces of mobile applications inaccessible from a web browser.

This thesis aims at describing its development process.

Contents

1	Introduction	1
1.1	Context of the Problem	1
1.2	Objectives of the thesis	2
1.2.1	Mission Statement	2
1.2.2	Working Hypotheses	4
1.2.3	Research Method	5
1.3	Overview of the Thesis	6
2	State of the Art	7
2.1	Key concepts	7
2.1.1	Beauty	7
2.1.2	User interfaces	8
2.2	Methods	10
3	Development	15
3.1	Requirements	15
3.2	Architecture	16
3.2.1	Overview - iOS application	18
3.2.2	Overview - Server	22
3.3	Mobile implementation	24
3.3.1	Home	24
3.3.2	Input preview	25
3.3.3	Input gathering	26
3.3.4	UI rating	28
3.3.5	Consistency metric	30
3.4	Aesthetic metrics implementation	30
3.4.1	Balance	31
3.4.2	Density	33
3.4.3	Equilibrium	33
3.4.4	Proportion	35
3.4.5	Symmetry	35
3.4.6	Consistency	37
3.4.7	Future work	37
3.5	Borders recognition model implementation	37
3.5.1	First attempts	37
3.5.2	OpenCV - first version	39
3.5.3	OpenCV - second version	41
3.5.4	OpenCV - final version	42

3.6	Testing	43
3.7	Download	43
4	Software validation by examples	45
4.1	Amazon	45
4.1.1	Home	45
4.1.2	Cart	48
4.1.3	Consistency	51
4.2	SNCB	51
4.2.1	Home	51
4.2.2	Purchases	54
4.2.3	Consistency	57
4.3	Comparing results	57
4.4	Conclusion	58
5	Final Conclusion	59
5.1	Overview	59
5.2	Future Work	59
5.2.1	Adding more metrics	59
5.2.2	Experimenting with different formulas	60
5.2.3	Octodollop as integrated feedback gathering tool	60
A	Server	65
A.1	Sample requests and responses	65
A.1.1	/rating request	65
A.1.2	/rating response	66
A.1.3	/ai/bounding_boxes response	67

Chapter 1

Introduction

1.1 Context of the Problem

It is estimated that people spend almost one third of their waking time using smartphones, either by browsing the web or using mobile apps. Mobile devices have nowadays become a natural extension of our arm, with the gesture of reaching for one's phone and unlocking it turning into something almost instinctive for a big chunk of the population.

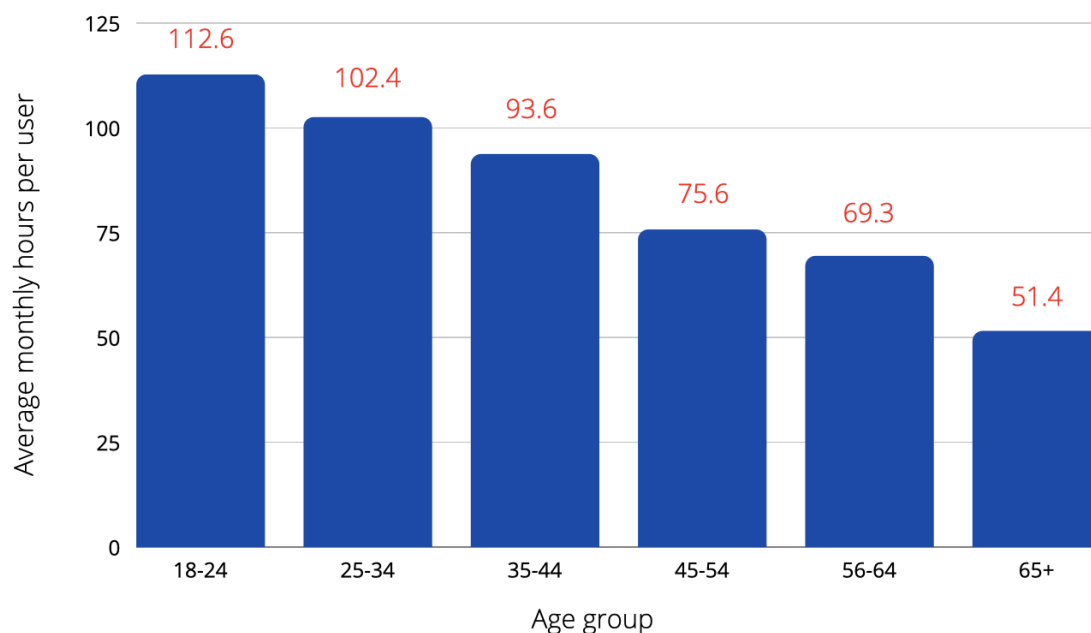


Figure 1.1: Average monthly hours per mobile-app-user by age

However, in contrast with the significant portion of their days allotted to mobile phone usage, it has been found that users spend the majority of their on-screen time only on a handful of applications: long gone are the days of self-made app millionaires, which were able to accrue small fortunes by publishing rough, unpolished apps.

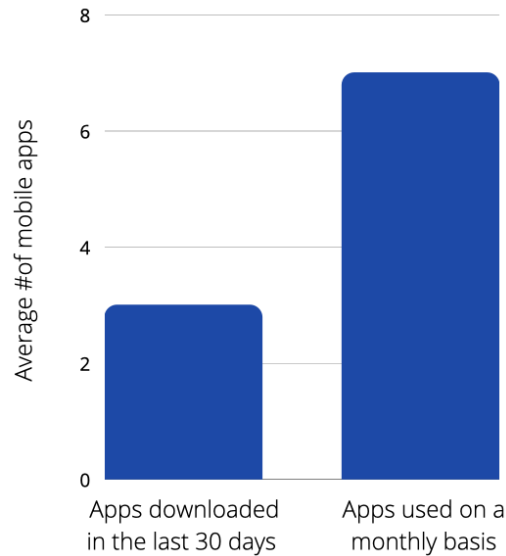


Figure 1.2: Mobile app usage

Nowadays, users seek nothing less than perfection. While this concept can be translated into more standard metrics such as reliability, responsiveness and usefulness, a major part of it, often overlooked by developers, is aesthetics.

The underplaying of this important aspect of the user's experience when interacting with an app, lead to the lack of tools and procedures to systematically and methodically measure the beauty of user interfaces (UIs). Procedures that which, when talking about other more common quality metrics such as the ones previously mentioned, are on the other hand tracked and monitored on a regular basis and used to have a clear idea of the evolution of the product itself.

If to this we add the fact that the conceptualization and refinement of UIs are often carried out by a figure external to the development process, namely Graphic Designers, which are generally less familiar with automated workflow tools, the entire idea of grading UIs aesthetics automatically and consistently becomes much harder to integrate in a user interface validation process.

For all the aforementioned reasons, having a structured, easily accessible method for evaluating the appeal of user interfaces becomes fundamental in order to retain as many users as possible, increasing the app's notoriety in the meanwhile. By making sure only highly rated UIs go into development and get pushed to production, a lot of the trial and error that is generally associated with user interfaces validation could be removed, reducing feature development time considerably and providing additional instruments to evaluate the evolution of the product's health under the aesthetics perspective.

1.2 Objectives of the thesis

1.2.1 Mission Statement

The purpose of this thesis is to examine possible solutions to enable a large audience, both with and without a technical background (developers, project managers, designers, ...), to evaluate the user interfaces of mobile apps.

Given the broad user-base that the tool will target, it is of essential importance for it to be easily and readily accessible. Various possible solutions have been explored. However, most of

them have been found too narrow in their usage, often requiring a specific skill-set that would necessarily exclude a large portion of the intended audience:

- *CI or Snapshot Testing tool*: while this specific solution could be of great value for developers, which could set up automatic triggers to run UI rating jobs periodically and get notified when certain aesthetics thresholds are unmet, this approach would practically cut off any part of the workforce with a non-technical background, with the added disadvantage of prohibiting the validation of the fitness of user interfaces before actually developing them, making this tool ultimately unusable during a product's validation phase.
- *Graphic design software extension*: an extension capable of integrating with major UI design software (Adobe XD, Figma, Sketch, ...) could certainly be helpful for graphic designers, who generally spend a good portion of their time on this kind of programs, but it would be quite burdensome to use for other people not familiar with those tools. Not to mention that most design software packages require a paid subscription, posing an additional entry barrier for anyone looking at getting their app's UI graded.
- *Website*: certainly getting closer to our requirements, being websites usable by the entirety of our target audience, they still aren't entirely ideal when it comes to mobile user interfaces in particular. As the UI rating process will foresee-ably include high-precision actions such as dragging, zooming and long-pressing, having to do all this on a mobile device might result in quite an uncomfortable experience. While one could always export screenshots taken on a mobile device to their computer and access the website from there, this poses an additional, avoidable step. Thinking there could still be room for improvement, this solution was discarded as well.

After careful consideration, it has been taken the final decision of **developing a mobile application capable of outputting a quantitative and qualitative measure given a user interface as an input**.

While this solution provides all the availability advantages already explored when considering the "Website" option, it also addresses its shortcomings by providing responsive gesture recognition by leveraging the native development environment, on top of added independence, autonomy and portability.

These last mentioned points were amongst the driving factors that further on corroborated the choice of a mobile app as one of the possibly best options for a user interface rating platform. A bootstrapped evaluation engine, such as the just described adopted solution, makes the most out of the hardware and software working together with the objective of solving a common problem by performing a series of actions. The developed application will in fact be able to gather data about the user interface (the UI building blocks), find a solution based on this information (the algorithms used to compute the various UI metrics) and solving a problem (displaying such results) without the aid of any external service, machine nor platform.

It then follows that the final tool will be highly portable as well, with the end-user practically owning the evaluation engine itself by being able to have it with him at all times, as it resides on the user's own smartphone.

All these characteristic make, for the purposes and objectives that were initially detailed, a bootstrapped approach to be preferred over non-bootstrapped ones. These would in fact only cause the whole evaluation procedure to be inherently disruptive and disrupted: by forcing the user to switch between different, potentially non-integrated contexts, the entire UI rating engine would lose not only in immediacy, but in robustness and availability too. Having many distinct working components, on which the developer might not even exert direct control over, only adds to the risk of having a lower overall availability, as a tightly coupled integration between them

would cause the entire system to go down if one were to fail. Cohesion, while not technically impossible, would still significantly be hindered due to the notable development efforts resulting from the added porting interfaces and infrastructures needed in order to make the many moving parts of a non-bootstrapped system communicate between each other.

While robustness would still be achievable by, for example, development through a modular monolith architecture or backing fail-safe microservices, the resulting product wouldn't still be as sound as a bootstrapped one.



Figure 1.3: The logo of the mobile application, Octodollop

Throughout this thesis, the terms "the application", "the app", "the tool" and "Octodollop" will be used interchangeably.

1.2.2 Working Hypotheses

Throughout this thesis, the following working hypotheses have been assumed:

- H1.** *Device and platform specifications:* the UI rating tool should be easily and readily accessible by anyone through their mobile devices. Given the wide range of smartphone brands, their different capabilities and performance differences, rather than making a cross-platform app capable of running on a wider range of mobile operating systems with sub-par performance and with higher chances of unexpected behaviours, the app has been developed for Apple devices (iPhones and iPads) exclusively, using Swift to leverage the native development environment to ensure a top-notch, reliable and fluid user experience.
- H2.** *Environment specifications:* the app will be used on Apple mobile devices to grade user interfaces provided in the form of screenshots. The screenshots are meant to be captured by the user itself, outside of the app's lifecycle. While originally attempts have been made to integrate OctoDollop in the screenshot capturing process, with the initial intent of "injecting" some of the tool's behaviour in the app to rate, these possibilities have quickly been abandoned due to Apple's privacy requirements and restrictions, which, if circumvented, would have made the app ultimately non-publishable. Exceptions have been made for websites publicly accessible through standard web browsers, for which an in-app capture mechanism has been provided.
- H3.** *Target audience:* the tool is aimed at a broad, general audience with no particular domain-specific knowledge but with basic tech-literacy, capable of operating a mobile device and navigating a mobile application intuitively and with ease.
- H4.** *Aesthetic metrics:* for the computation of the beauty rating itself, it has been decided to use the same aesthetics metrics mentioned in the work "Assessing User Interface Aesthetics based on the Inter-subjectivity of Judgment" [17], as they have been found to have positive statistical correlation against how humans perceive the attractiveness of UIs. In our case, the metrics have been further on grouped into categories and aggregated into

partial results, in order to offer a better overall picture to the end-user by enabling him to more easily and rapidly interpret the results.

1.2.3 Research Method

The research method used during the course of the thesis is assimilable to the "verification & validation" model. After an initial validation procedure, where a system meeting the real objective's needs was designed and structured, the development activities were carried out, followed by various stages of verification processes, in which the conformance of the software to the specifications was checked against and, at times, lead to the initial plannings themselves being slightly modified, refined or adjusted.

While other software development techniques have also been taken into account when considering the appropriate research method to employ, their usage was ultimately decided against.

The DevOps approach, while it leverages Continuous Integration and Continuous Development workflows to ensure the constant release of new updates and fixes to production, wasn't in our case deemed necessary due to our entirely non-existent user-base, technically only comprised out of end-to-end and acceptance testers. Because of this, in addition to the relatively small initial set of functionalities offered by Octodollop, the need of publishing new features on regular, small intervals of time was not something that was needed, having the only real effect of significantly increasing development times due to the notable infrastructure overhead that would have burned the project in its very early stages.

Agile development practices, which generally benefit teams thanks to the iterative, feature-based methodology, wasn't really zealously applicable in our case, due to the app's essentially consisting of a single core functionality: UI rating. Even though this could have technically been broken down into smaller steps, the great workload unbalance related to each one of them wouldn't have really made this work style ultimately fruitful.

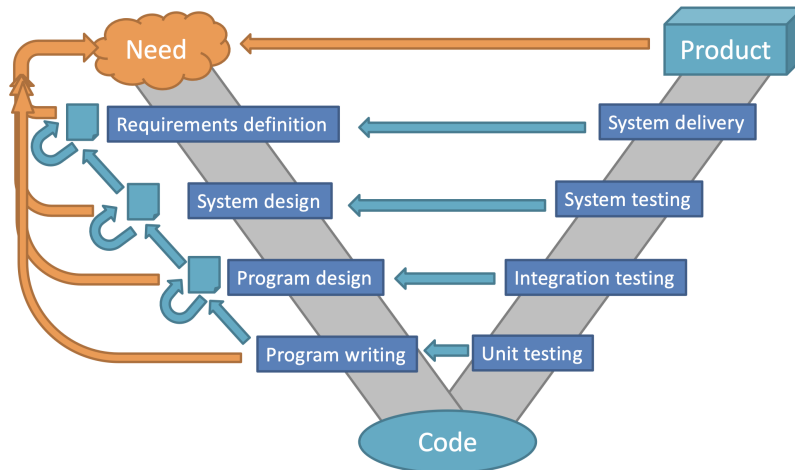


Figure 1.4: Sequential phases of the validation & verification model

During requirements definition, the possible solutions employable to allow the end-user UI grading capabilities have been explored, with additional focus on the subset of aesthetic metrics to compute in order to provide meaningful and complete user interface ratings.

In the system design phase, the overall structure of the app, the system's architecture and the interaction between the client and server have been defined.

During the program design and program writing, the platform-specific languages and frameworks to be used were chosen, focusing on the realization of a high level class interaction diagram as well.

The implementation then followed according to the specifications established during the previous steps.

The testing and delivery parts of the process won't be described in detail during this thesis and are to be considered out of scope, with only a brief mention of the unit tests added to the system in 3.6. Additionally, Quality Assurance activities such as code inspection, and acceptance testing have been carried out throughout the entire development process.

1.3 Overview of the Thesis

To reach the central objective assigned to this thesis, the manuscript is structured as follows.

Chapter 2 explores the main concepts around this thesis, analysing past literature with the objective of reconciling the concepts of beauty with that of user interfaces.

Chapter 3 goes over the implementation details of the entire application infrastructure, both on the mobile and server side.

Chapter 4 puts the concepts and notions described in the previous chapters into practice by using Ocotodollop to rate two mobile application UIs.

Finally, Chapter 5 sums up the contributions of this thesis and offers some possible ideas for ramifications of future work.

Chapter 2

State of the Art

2.1 Key concepts

In this section, the two core concepts of this thesis will be explored: beauty and user interfaces.

2.1.1 Beauty

Throughout the centuries, beauty has been a value sought after and aspired by the many. In plentiful works of art, their conception bore from nothing more than the desire of capturing small glimpse of beauty: human beauty, natural beauty, decadent beauty... the facets of this core concept in human existence are innumerable, each carrying its own significance and set of emotions, but all linked by a common thread.

No matter its origin, in fact, humans are naturally attracted towards beauty. Sparking from a sensation of pleasure caused by a perceived sense of harmony and balance, the feeling then expands and insinuates into our minds in the form of appreciation and, ultimately, peace. [18]

Although in recent years some new social movements have shown adersion towards the concept of beauty, de-glamorizing it by stating that people shouldn't base their judgment solely on the outer appearance of things ("Judging a book by its cover"), research shows that the reality of things is actually quite different: extroversion can be accurately perceived after only a 50 milliseconds exposure to a face, just like dark triad aspects of one's personality (Machiavellianism, narcissism and psychopathy) [15].

In other psychological trials, participants were tasked to determine personality traits of individuals based off on nothing more than their photo. The results showed that the more aesthetically pleasing people were assumed to possess more socially desirable personality traits and were thought to lead more successful lives than their unattractive counterpart [6].

As further proof of the deeper implications of these findings, which one might erroneously disregard as simple first impression biases that the rational mind will eventually rule out, the ever-occurring effects of the natural attraction of mankind towards beauty have very tangible consequences in our medium and long term decisions as well, with, for example, job seekers being regarded as more beautiful than others having a significantly higher chance of being hired as well [16].

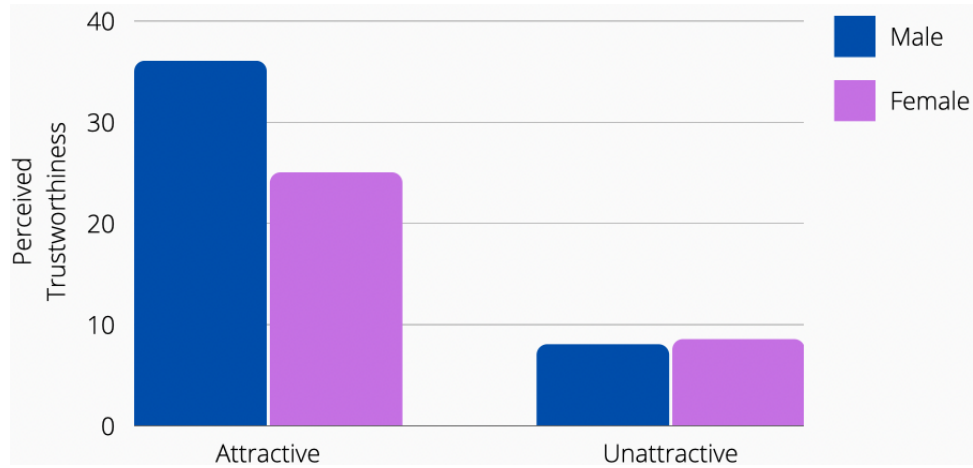


Figure 2.1: Perceived trustworthiness of people based on their beauty (adapted from [8]).

The evidence of a clear bias towards beauty is apparent from the research and literature of the last centuries and it would be foolish to assume it not extending to modern mediums as well. Hardware devices, webpages, mobile apps: no matter the source, the utility, the quality nor reliability, human beings will always be influenced by the unavoidable, almost innate, feeling that "What is beautiful, is good".

Ignoring such fact, will lead to the ultimate, maybe unjust, "death" of the product in question.

2.1.2 User interfaces

We could define user interfaces (UIs) as the main, sometimes single, entry point of an application's features. UIs can generally be classified into three main categories:

- *Graphical user interfaces (GUIs)*, in which the user interacts with visual representations of interactive and/or informative building blocks laid out with the objective of conveying a message. The interaction generally takes place through a tactile input device (mouse, keyboard, touchscreen, ...).
- *Voice-controlled interfaces (VUIs)*, distinguishing themselves from GUIs from the way a user interacts with them: action triggers and information retrieval procedures happen in fact exclusively through voice commands. Examples of VUIs include Siri (Apple), Alexa (Amazon), Google Assistant (Google).
- *Gesture-based interfaces*, mainly revolving around experiences like Augmented Reality and Virtual Reality, thanks to which the user is able to interact with a 3D space through bodily motions and three-dimensional gestures and movements.

As this thesis exclusively focuses on graphical user interfaces, being the other two categories out of scope, the term GUI and UI will be used interchangeably.

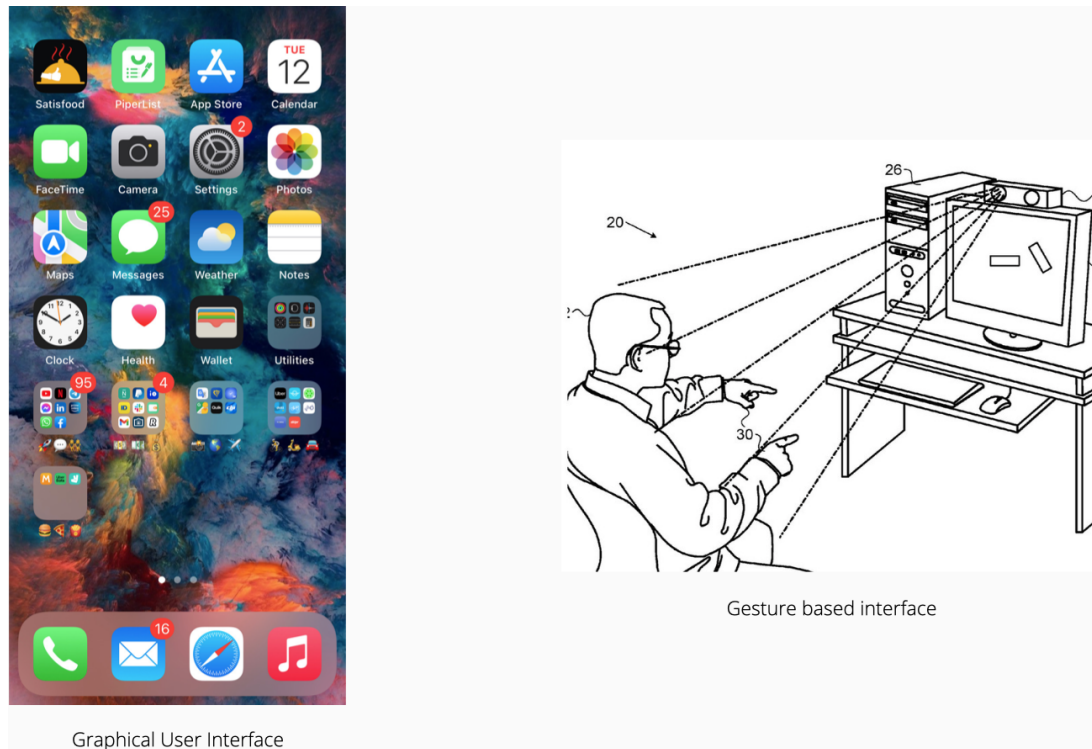


Figure 2.2:

Left: the graphical user interface of the iOS operating system

Right: a gesture-based interface, in which the user gestures are recognized and captured through the aid of a camera attached to a computer

With the help of UIs and their explicit and implicit visual cues, users are facilitated in carrying out the specific tasks they wish to complete when using an application. Thanks to the employment of icons, text labels, input fields, buttons and other base components, developers are able to convey complex messages and instructions in a quick, intuitive, platform-agnostic way.

In the past, the user's only way of interacting with a computer was through a set of explicit, textual orders, either with a predefined set of commands provided by the running application itself or directly with programming languages. This often lead to a very large, complex instruction set which could vary from platform to platform or, worse, from application to application.

In today's age, latest UI design trends and best practices have shifted their attention to providing a consistent user experience no matter the system or software being used, leveraging the familiarity that users have built with iconography throughout the decades in order to flatten the learning curve needed in order to get familiar with a new piece of software. [2]

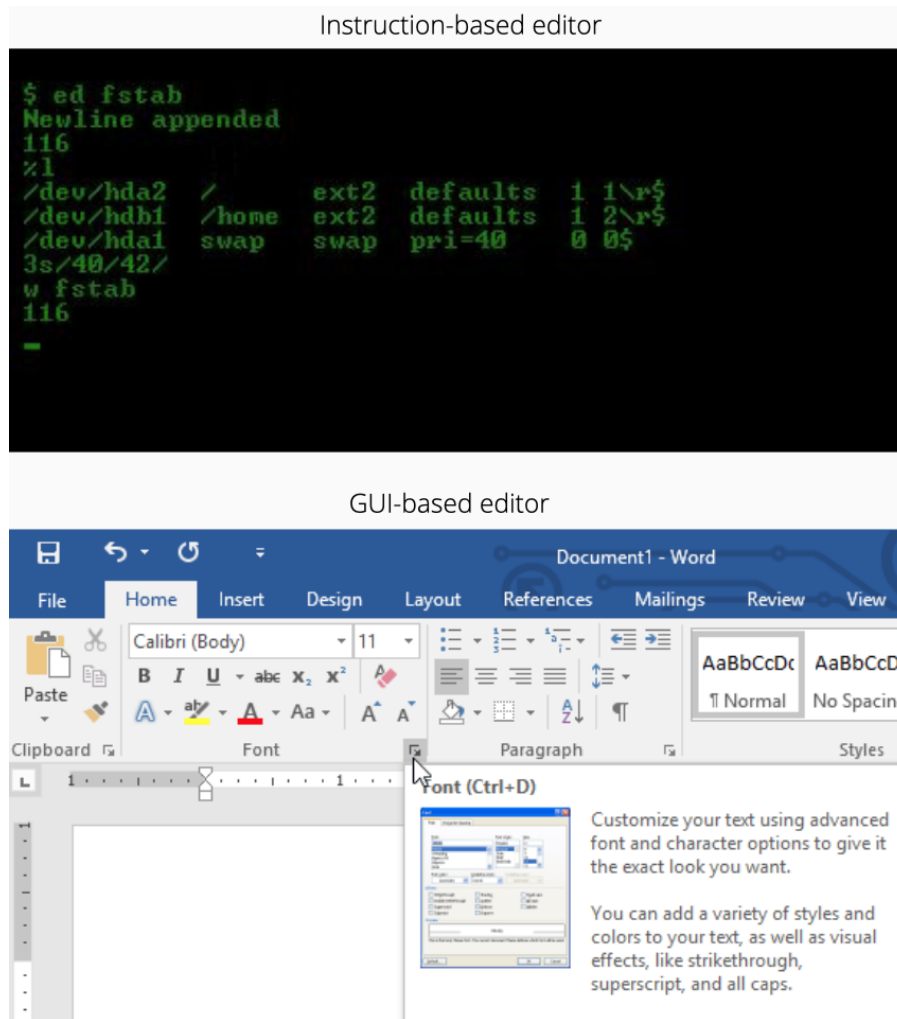


Figure 2.3:

Top: ED, an instruction-based text editor released in the 1970s

Bottom: Microsoft Word, a widely used, GUI-based, modern text editor

It then becomes clear that UIs massively helped the widespread of computers, and user-facing software in general, by lowering the entry barrier needed in order for people to interact with these kind of complex, elaborate systems that software applications are. Thanks to them, operations that only a selected few were able to carry out a few decades ago, like documents editing through word processors and cash flow handling with spreadsheets, are now an everyday deal that a good chunk of the population is able to comfortably execute.

2.2 Methods

After having analyzed the concepts of beauty, user interfaces and how they are related to our study in question, it is now within our objectives to understand how the two overlap, interact with each other and how one can be used to improve the other.

As it was already previously mentioned, beauty is something that touches and has touched us in nearly every aspect of our lives, ever since the dawn of times. It would be naive to assume that this value doesn't extend to modern mediums and high-tech creations as well: the shape of a monitor, the material of a phone, the layout of user interfaces... all these things, and more, are heavily influenced by human perception of beauty, which, in many cases, can be a driving

factor for sales, purchases and downloads.

When talking about man-made beauty, the first things that generally pops into people's mind are paintings, sculptures, photos. Artistic creations in which, by the careful placement of subjects and elements, a sense of harmony and balance is reached and transmitted to the viewer. But aren't user interfaces artistic creations too, after all? Don't graphic designers execute the same kind of tasks as painters and sculptors? The parallelism is quite straightforward, UI designers ("the artists") create user interfaces ("the paintings") through the aid of the appropriate software suite ("the brush and canvas") seeking the same values that so many of their century-old predecessors did: equilibrium, balance, proportionality... beauty.

The links between beauty and UIs are profound and deep rooted. For this reason, seeking ways of applying beauty principles and metrics to user interfaces is of primary importance to be able to provide premium user experiences.

Multiple experiments and studies have been conducted regarding the specific facets of beauty that influence the perceived attractiveness of a user interface, here follow some of the most notable case studies.

Launcher, a Visual Basic application, analysed complexity as a specific trait of UI aesthetics, trying to link it with the overall software usability. [5] Although design manuals generally support the idea of placing elements according to a grid pattern, the data collected through this tool wasn't able to support this widely accepted rule, showing on the other hand that user interfaces with low complexity, which were more closely laid out according to a grid fashion, actually gathered lower usability ratings.

In Questim, another study on the matter conducted at UCLouvain, a web tool capable of giving a rating to web pages was developed. [17] By taking into account a large plethora of beauty aspects, such as balance, density, alignment, concentricity, simplicity proportion and symmetry, the researchers were able to highlight specifically the metrics of balance and simplicity as good predictors of user's attractiveness perception of UIs.

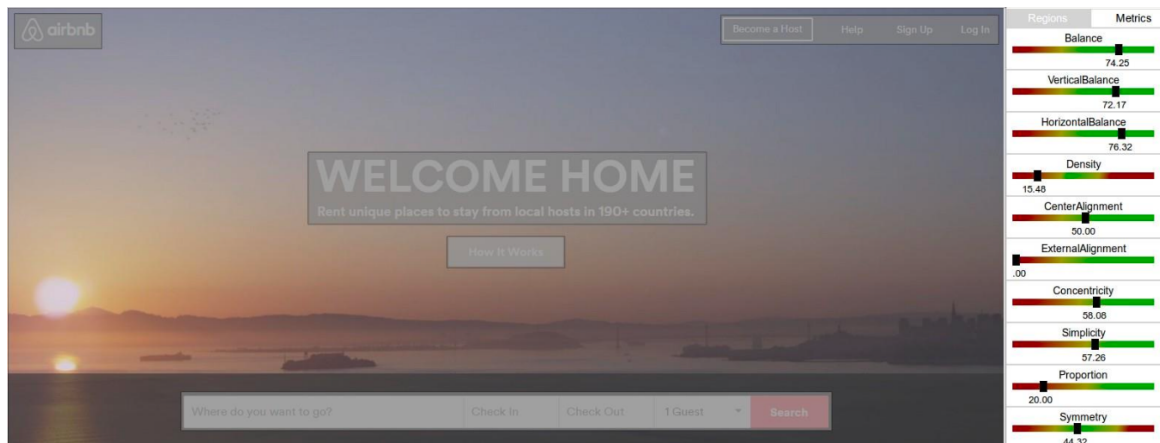


Figure 2.4: The results of the aesthetic metrics computed by Questim on the Airbnb website

Through USAGE, psychological models were brought into the evaluation of applications' cognitive analysis [4]. To be operated at a low level by the developer itself, by linking user actions to specific app components directly in code, USAGE, given a list of user actions expected in order for a task to be carried out successfully, can then output an estimate of the time need in order for the specified activity to be completed by the user. The principle behind this obtained temporal measure is that tasks that take long amounts of time to be finalised by the user might be indicative of badly designed user interfaces.

Miniukovich tried to reconcile automatically generated UI ratings with user interface preferences expressed by real users in a lab environment [10]. By taking into account visual clutter, color range, number of dominant colors, figure-ground contrast, contour congestion, grid quality and white space, the study showed the metrics were able to estimate with fairly accurate precision, when it came to web pages, human preference towards certain screens rather than others. The results weren't as promising when it came to mobile applications though, suggesting that more accurate research ought to be conducted taking into account the application's genre and category into the computation of the aesthetic metrics themselves.

In AIM, AAlto Interface Metrics service, attempts have been made on computing the beauty of web-based user interfaces focusing mainly on colors and the user navigation experience itself [13]. Dividing each employed metric into either "Colour perception", "Perceptual fluency" or "Visual guidance", AIM enables the breakdown of a screen into constituents that go beyond its mere appearance, taking instead into account factors such as "The mental effort needed to differentiate spatially proximal lines" and "The predicted human attention on different design classes and natural images", incorporating effectively user experience into the factors to weigh in when assessing the fitness and attractiveness of a UI.

Summary

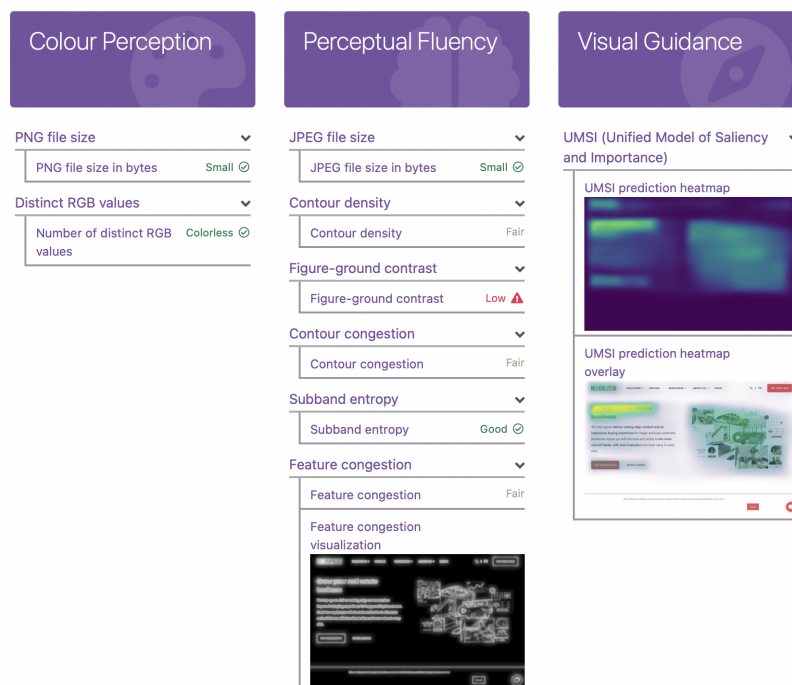


Figure 2.5: An example of the results obtained through the AIM platform

SAM, unlike the studies we have explored so far, doesn't focus on user interfaces as a whole, bringing the spotlight over a nearly-ever-present component in all web pages: navigation menus [7]. By enabling the developer to specify target policies for individual menu items and

adaptation policies for how the items themselves should be reordered based on their scores, SAM truly incorporates the concept of "Beauty is in the eye of the beholder" into its framework, enabling the realization of user interfaces dynamically customizable by the end user.

From what we have seen, the choice of the metrics to evaluate when giving aesthetic ratings to UIs is very important, as some seem to have a tighter connection to human indicators of beauty than others.

For example, Ngo et al. were able to demonstrate close correlation between their devised aesthetic model and the actual perceived attractiveness of user interfaces [12], measuring many aspects such as balance, equilibrium, sequence and complexity. The authors then go on about suggesting that the experimental results by them obtained could be further on improved by expanding the range of beauty facets by them considered, accounting, namely, for the distribution of colors and font weights.

Ngo further expanded his study in 2001 [11] by adding the metrics of unity, proportion, simplicity and economy in an effort of grouping together all the relevant aspects that should actively be integrated, or attempted to be, into design tools with the objective of empirically and consistently designing better user interfaces.

In research conducted by Ataboli et al. [1], through the use of a regression model calibrated on the joint usage of the balance, unity and sequence formulas on top of previously conducted lab studies, it was shown that the three aforementioned metrics were very good indicators of user's perception of UI beauty.

A peculiar evaluation metric found in the SHERLOCK software consists in spell-checking, terminology concordance and dissonance metrics, suggesting that inconsistent use of key terms and phrases causes user's performance to slow down between ten and twenty-five percent and, as a result, a drop in the perceived attractiveness of a visual piece of content [9]. The results were statistically significant, although, being the study conducted in 1997 and being nowadays' user interfaces no longer predominantly comprised out of text, potentially no longer as relevant as they might have used to be.

Lastly, by leveraging the help of a large and diverse volunteers pool, Reinecke et al. were able to devise a model to compute perceived colorfulness and visual complexity of user interfaces from screenshots of website pages [14]. These metrics, if coupled with the demographic data such as users' age and education level, were then shown to explain almost half of the variance related to the first impression (fifty milliseconds exposure) of websites' attractiveness.

The current state of the art shows a significant correlation between some metrics used to measure the attractiveness of UIs and the actual human perception of user interfaces beauty, with evidence suggesting that more accurate predictions are obtained when jointly taking into account multiple facets of beauty, rather than considering them individually.

With the analyzed researches showing a predilection towards certain metrics as more reliable determinants of human judgment, a careful choice of the aesthetic aspects to take into account when computing a UI's appeal rating should allow us to generate accurate, forecasting results.

Chapter 3

Development

In this third chapter, the development process of Octodollop, the iOS application capable of computing UI ratings for user interfaces, will be explored.

After going over the overall project's requirements, the system architecture will be illustrated, both for the mobile application and its backing webserver. Implementation details will then follow.

3.1 Requirements

The product to be developed is a mobile application for iOS devices (iPhones and iPads) with the ultimate objective of, given as an input a user interface, either as a screenshot or as a URL redirecting to a webpage, outputting a comprehensive score for the UI in question, showing an overall beauty rating and its detailed breakdown into the various aesthetic metrics it is made out of. This behaviour, which to the user appears as self-contained in the mobile application, makes the evaluation tool itself extremely portable as a UI rating can be computed anytime and anywhere on the user's device in an immediate and straightforward way, consistently with the adopted bootstrapping approach.

The aesthetic metrics to be included in the final output must comprehend balance, equilibrium and symmetry, whose choice has been driven by the study [12] conducted by Ngo et al., density and proportion, following the results obtained by Questim [17], and, finally, consistency, taken by the research conducted by Burny et al. [3]. The metrics are to be computed on-premise without the aid of external services or APIs, in order to make the entire system as independent and autonomous as possible.

The meaning and formulas used for the computation of each one of these metrics will be explained later on in the chapter.

Before starting the rating process, the user should be given the option to decide whether the user interface should be taken from a website (URL) or from another mobile application. In the latter case, the user should be allowed to import a screenshot of the UI from his photo gallery. The screenshot capture process is to be considered carried out outside of the application's context and operational scope.

Once the UI has been selected, the user will then identify all the basic elements of the user interface. The identification process should be able to be carried out both manually and automatically, through the aid of border recognition artificial intelligence. The basic UI elements are to be detected exclusively as rectangles: other shapes or free-hand drawing should not be made possible. A circular button, for example, will then be identified as a square enclosing the perimeter of the button itself.

On completion of the identification process, the user should be able to see the ratings of the user interface, which will be dependent on the UI elements identified on the previous step.

The UI rating can then be discarded or saved. In the former case, the rating won't be retrievable anymore. In the latter, the user will be able to access the rating and see all of its related metrics at any later moment, with the additional option of using said rating, in conjunction with other saved ones, to compute consistency scores.

The saved ratings don't need to persist between app installs, nor be synced between different instances of the application.

3.2 Architecture

The iOS application has been developed entirely in Swift, with no external libraries or frameworks.

The app's architecture follows the Model-View-ViewModel pattern (MVVM), an event-driven architecture in which the ViewModel serves as a middle layer between the model and the view. With this pattern, the entire business logic is contained inside the ViewModel, responsible for both updating the model and the reception / redirection of events and data, accomplished in native Swift through the aid of closures, asynchronous blocks of code executable at a later time. The View, enclosed in ViewControllers, is solely and entirely responsible for the UI logic and has no direct interaction with the model.

Other architectures had initially been considered but weren't deemed as a good fit for the project. With a typical MVC pattern, for example, the risk of having huge controllers was thought to be too high due to some complex app interactions and mechanisms, such as drawing overlays on screen during the element identification process, the displaying of identified UI elements, the ratings saving procedure and the handling of some state transitions. While a clean codebase would have likely still been achievable following this approach, the effort and strategical planning needed would have been much higher, on top of increasing the overall system's complexity and decreasing code readability.

Another option that was thought about was reactive programming. This approach is however not natively possible in Swift and would have required external libraries such as RxSwift. As any kind of supporting pods or framework have been avoided with the intention of making the project as easily maintainable and expandable as possible, even by other people external to the initial development process of Octodollop, this option was discarded as well.

The use of coordinators, in what would have effectively been an MVVM-C pattern, to better handle app's navigation and avoid ViewControllers dependencies, was ultimately not considered overly useful for the project's health in its current state, due to the small handful of screens and transitions present. In case the app were to grow considerably on future iterations, a migration to MVVM-C would still be possible and relatively easily achievable.

The UI layer of the application has been realized with UIKit, chosen over its newer SwiftUI counterpart due to its higher reliability, ease of manipulation and more extensive API.

The backend of the application, a server responsible for the computation of all the aesthetic metrics and for the border recognition AI model computations, has been built in the form of a REST API in Python using the Flask framework. With REST, the client uses GET, PUT, POST and DELETE HTTP calls to interact with the server. All network transactions are stateless and independent from one another.

The choice of Flask was in this case preferred over its full-stack counterpart Django due to its lightweightsness and expandability. Because of the very small number of endpoints that were going to be foreseeably developed in the first iteration of Octodollop, jointly with the fact that no display logic was to be carried out by the server nor no database interaction were going to be needed, Flask was considered to be a better option as it contained nothing more than the necessary functionalities, which could have very easily been expanded and adapted in case the

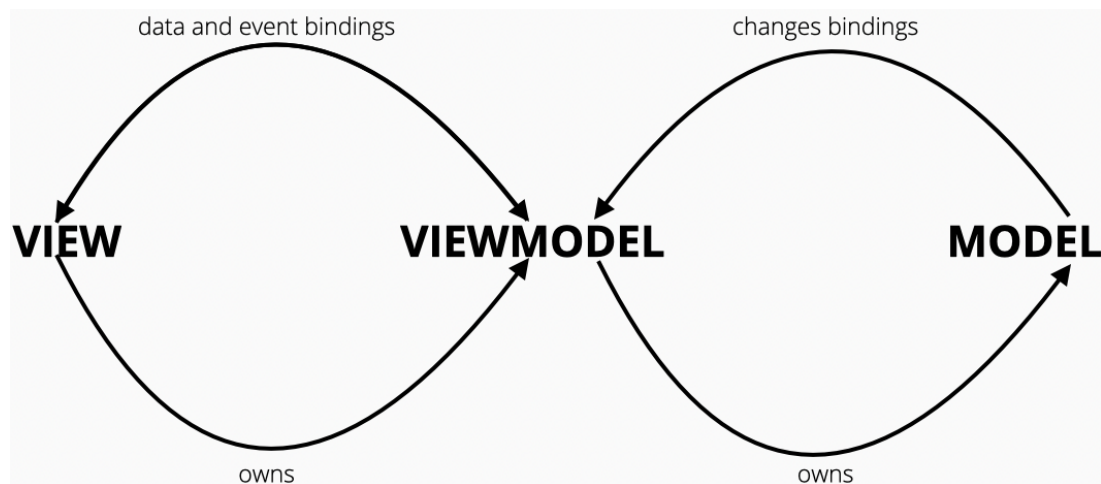


Figure 3.1: The interaction graph of the MVVM architectural pattern

server's needs and responsibilities were to grow in later versions.

Flask was also chosen over other python backend frameworks such as FastAPI due to its more widespread use and better support.

The server has been deployed on Amazon Web Services (AWS) EC2, chosen over other cloud services providers due to its competitive pricing, which allowed the project to be hosted and deployed completely for free, high availability and extensive support and documentation. EC2, on top of the aforementioned reasons, was in this case preferred over a more serverless approach, such as a combination of AWS API Gateway and AWS Lambda, thanks to the ease of development and deployment given by having access to a standard OS environment, which perfectly and seamlessly integrates with Flask. Going serverless would have required extensive setups and configurations that, for the use of this application, were deemed unnecessary even when considering the benefits of the potentially infinite auto-scaling offered by AWS serverless architectures.

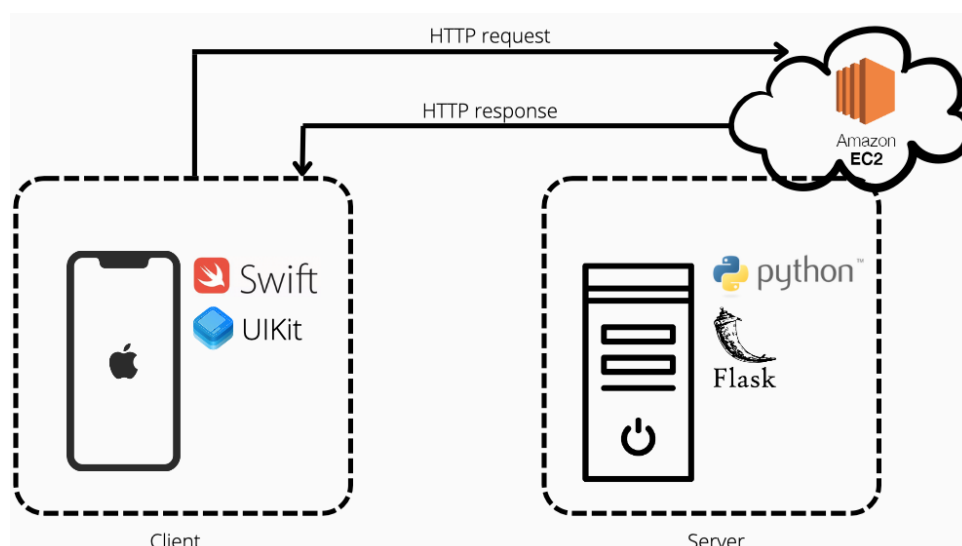


Figure 3.2: The overall system's architecture

3.2.1 Overview - iOS application

When opened, the application's initial state is in **Home State**, in which the user is asked for input in order to start the UI elements identification process. From the **Home State**, the user is additionally able to move to **History State**, where all the previous saved ratings are displayed and made accessible; backwards navigation from this point leads back to the application's entry point. Accessing a saved rating details leads to the **Rating State**, whose dismissal pops back to **Home State**.

Overview State poses the beginning state for the element identification process: the user can here see an overview of all the detected building blocks relative to the UI fed as input, starting off as initially empty. In case the user chose a website (URL) as the input source, a preliminary **Screening State** serves as a transition step, allowing the user to put the target webpage in the conditions he prefers. Afterwards, the webpage will then be "frozen", being converted into an image behind the scenes, converging into **Overview State**.

From **Overview State**, the user is able to quit the entire procedure altogether, reverting in this case to the initial **Home State**, or to identify UI elements, either manually or through the aid of an artificial intelligence. When the identification procedure is prompted to being, a transition to the **Identification State** is made, which can only be backwards navigated to its original presenter **Overview State**.

Once the user is happy with the number of UI elements identified, he can submit the user interface for processing. After the aesthetic metrics computation is completed, a transition to **Rating State** will be executed, displaying all the details relative to the UI beauty rating.

Additionally, if the user has more than two UI ratings saved, he can compute the consistency metric between two or more of them from **History State**, triggering a transition to **Consistency State**, in which the metric will be both computed and displayed.

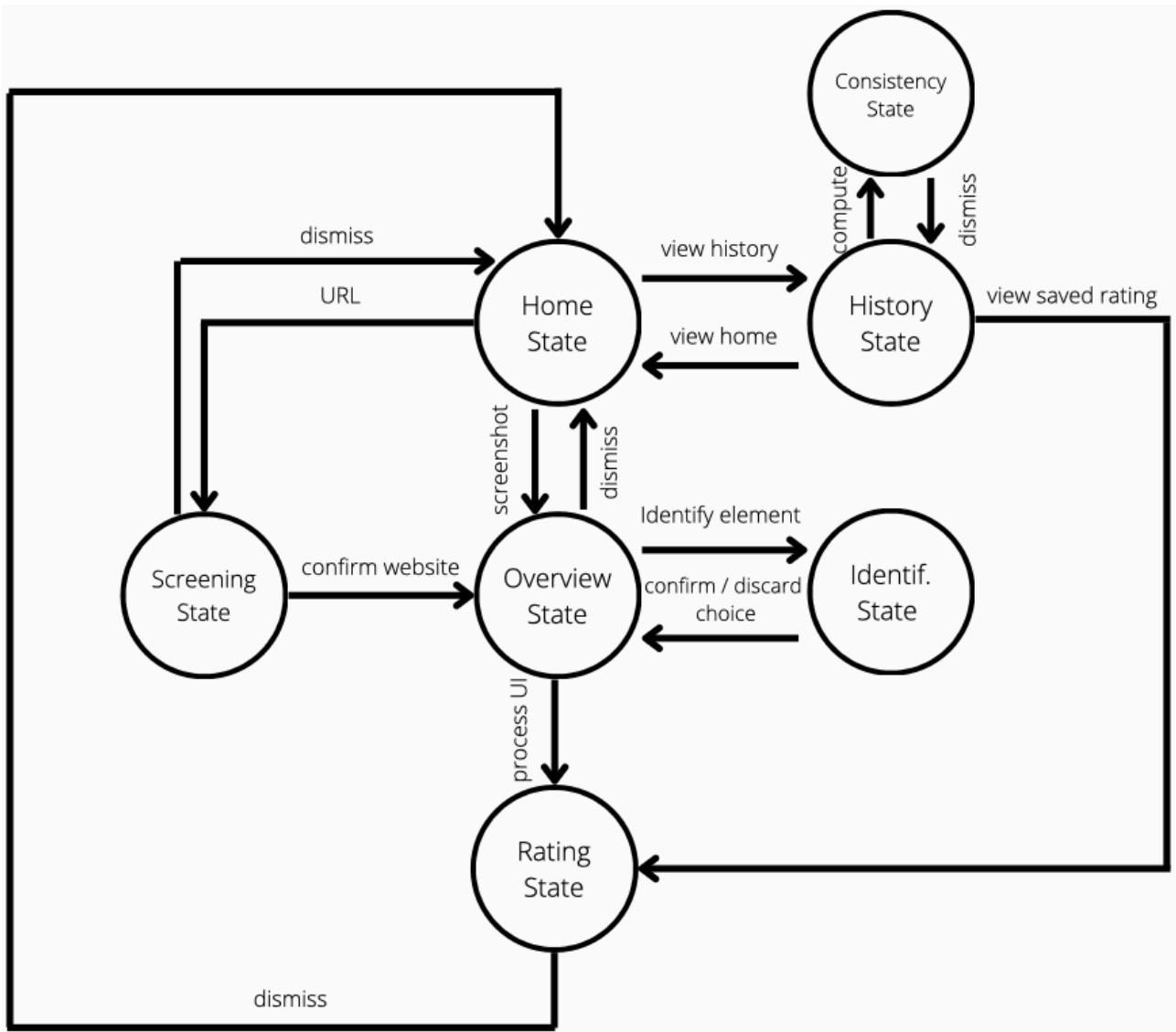


Figure 3.3: The Moore Finite State Machine representation of the iOS application

Going more into the implementation details, the concept of **Home State** has been translated into a **HomeViewController** wrapper, able to instantiate objects of type **ScreenInputViewController** and **HistoryViewController**. The former is where the actual user's input is gathered, while the latter is responsible for displaying and retrieving the saved user ratings from the local database (**History State**). The need of a wrapper is born from the navigation between the two states being triggered by a UI switch, persisting in an ever-visible and accessible context outside both view controllers, therefore belonging to the parent **HomeViewController**. In practice, **HomeViewController** is responsible for switching context between the input and history section, displaying all the UI capable of making the switch itself happen.

The **Overview State** and **Screening State** have been condensed into a single component: **InputPreviewViewController**. Due to the similarity of the two UIs and their core logic behind it, paired with the extreme simplicity of the **Screening State** functionality, consisting almost entirely of a **WebView** whose behaviour is entirely defined by default by **UIKit**, having a single **ViewController** with buttons that got progressively shown or hidden based on **ViewModel** notifications was, in this case, a good choice. The initial state of the component is determined by the initialiser used to instantiate it.

The methods related to the **Identification State** are split into two components depending on the identification type. The artificial intelligence aided procedure is contained entirely in **InputPreviewViewController**, therefore grouped together with the **Overview State**, as that consists of a single button tap that offloads all the image processing work to the app's server through a network call. The manual identification is on the other hand managed by **InputGatheringViewController**, requiring this step more complex interactions completely independent from the "Overview logic" and better suited for a new component.

The UI ratings results are handled entirely by **RatingViewController**, while the consistency metric is taken care of by **ConsistencyRatingViewController**, being this specific metric locally computed and requiring a different input gathering process than the one needed to get the overall UI rating. Both of these components have a 1:1 mapping with the functionality of their state, respectively **Rating State** and **Consistency State**.

A few supporting classes, external to the application main logic, have also been used (fig. 3.5).

NetworkManager takes care of the concrete implementation of network requests forwarding, parameters encoding and response decoding. No specific set behaviour has been defined for any of those steps, which get dynamically configured based off the **DataRequest** passed along as a parameter to the **makeRequest** method. The **DataRequest** protocol defines computed properties for the url endpoint, HTTP method, headers, body data and a function to handle responses decoding. The implementing classes (or structs) will then be responsible for defining each endpoint-specific behaviour.

DataManager handles the interactions with the local database instead, using the appropriate methods to encode the app's models into JSON-representable data (saving) and decoding them into their corresponding class implementation when needed (retrieving).

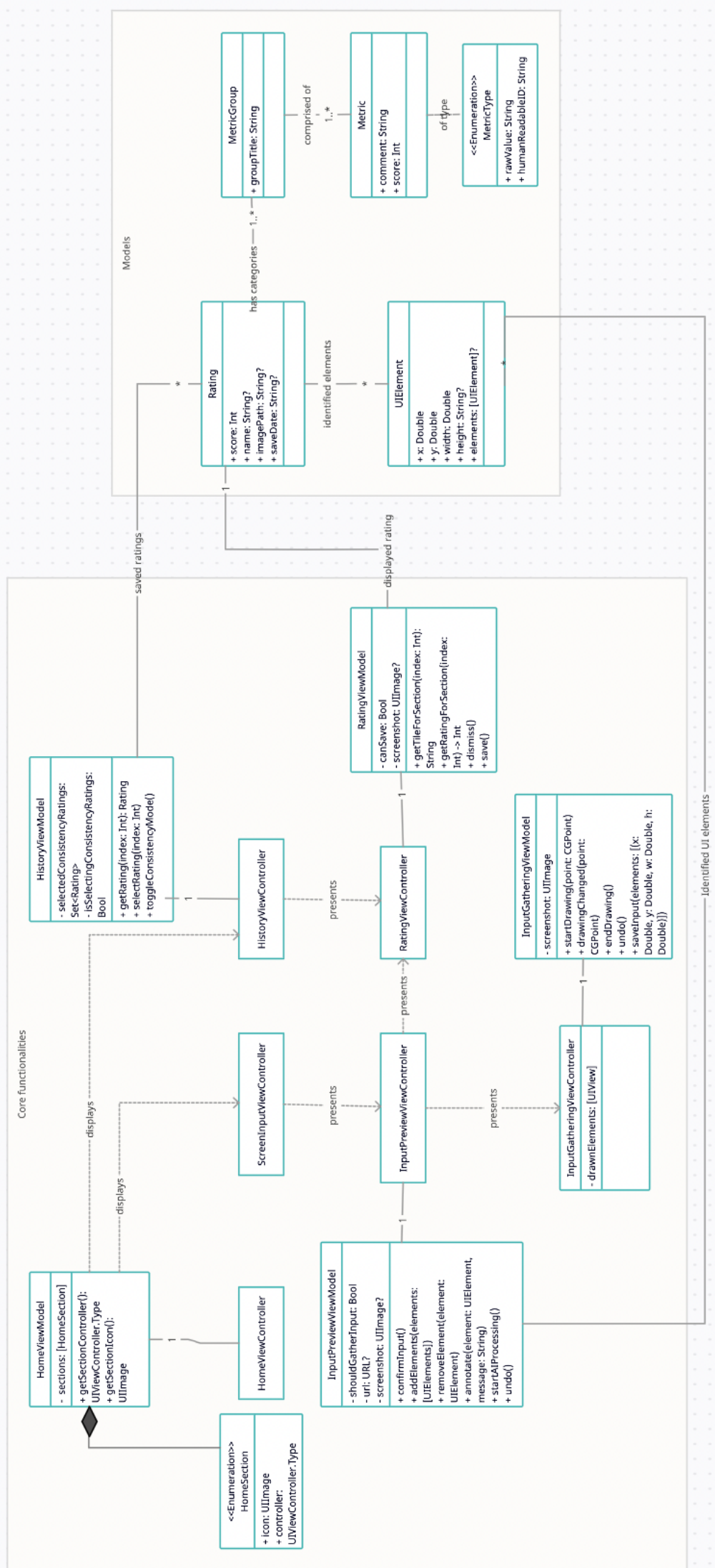


Figure 3.4: The class diagram of the core application's functionalities

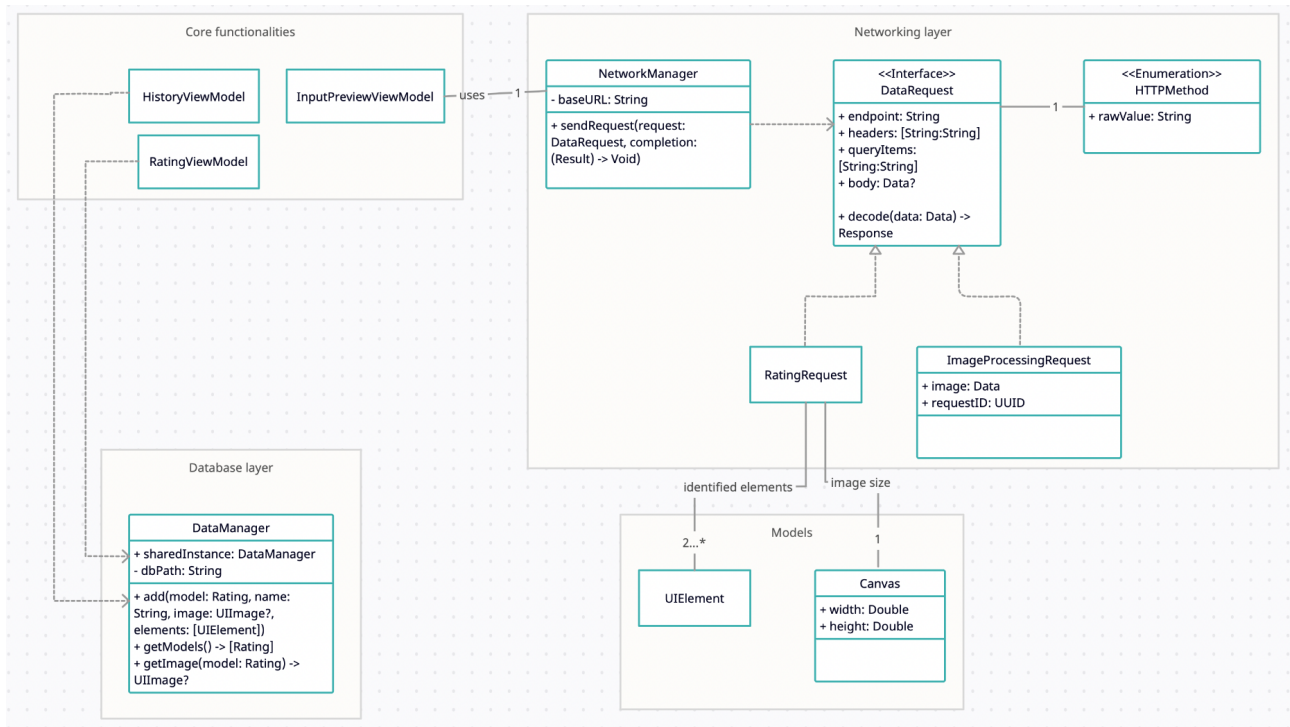


Figure 3.5: The class diagram of the application's helpers

3.2.2 Overview - Server

The server's entry point is in `octodollop/__init__.py`, in which the Flask server configuration takes place. The configuration can be divided in two steps:

1. Folder creation, in which the operational server's folder are created. These, on top of the default server's instance path, include the "upload" folder, where the screenshots uploaded for image processing will temporarily be saved.
2. Route setup, where all the server's API endpoints are imported and made accessible through public URL routes.

There are two available endpoints: `/rating` and `/ai/bounding_boxes`.

The `/rating` endpoint, given as input the data relative to the identified UI elements, computes and outputs all the available aesthetic metrics. The expected request should follow JSON syntax and rules, the response will be similarly sent in a JSON format. Any malformed or corrupted input will be met with a "400: Bad Request" response, no additional error handling or recovery attempt is made.

In the sample request at [A.1.1](#), the "items" field will contain all the identified UI elements, while the "canvas" field the absolute size of the user interface screenshot. All the UI elements are expected to be expressed with their relative positioning and sizing and should therefore be bounded between 0 and 1, as there can't rationally be an element bigger than the canvas itself, with negative length or positioned outside of the canvas. This is assumed to be a precondition satisfied by the client and no active checks are made to ensure that the values passed along respect these constraints. If a value outside of the operational range is given, the computation will continue as normal but the outputted results might contain inaccuracies.

In a sample response, an example of which can be found at [A.1.2](#), metric scores are integers between 0 and 100, their accompanying comments have a few words about the interpretation of the metric itself and its achieved score, while the overall rating is given as the mathematical average of the individual metrics.

The `/ai/bounding_boxes` request only takes as input a png image uploaded by the client as a "multipart/form-data" file. After having sanitized both the uploaded file and its name assigned by the client using Werkzeug utilities functions, the image is saved in the server's upload folder and processed by an OpenCV-powered algorithm, which identifies the UI elements and sends a response as a JSON payload having a structure analogue to the one illustrated in [A.1.3](#). The response lists in an array all the elements identified by the computer vision algorithm, expressing their positioning and size in relative coordinates similarly to the `/rating` request template.

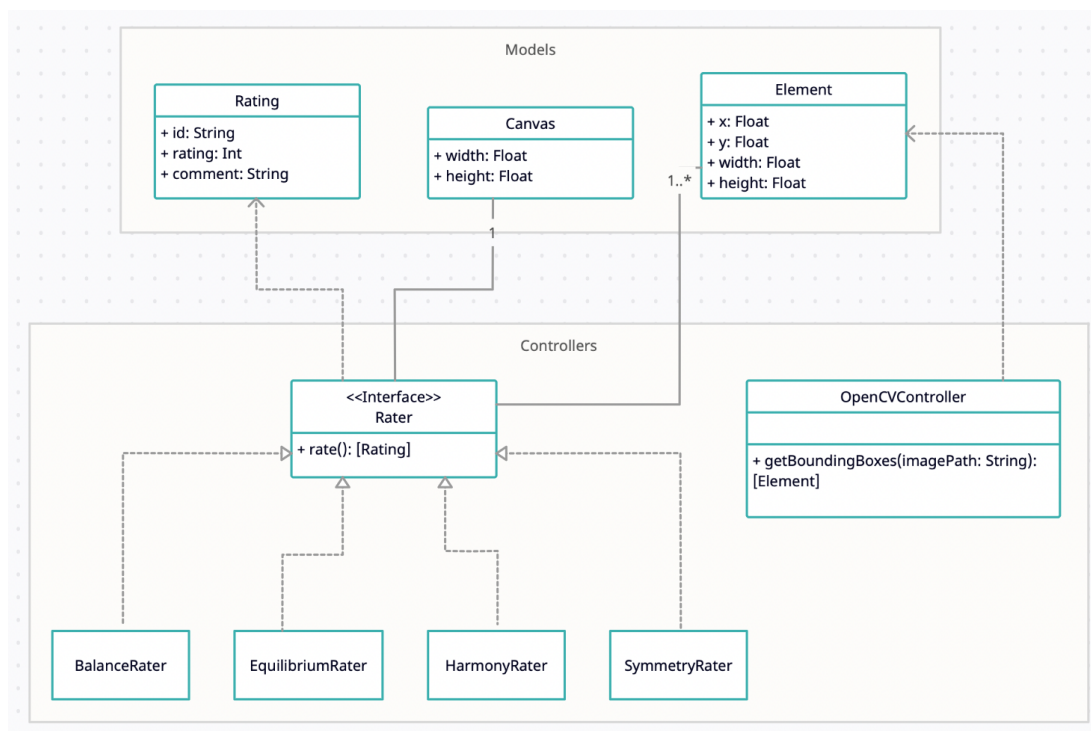


Figure 3.6: The server's class diagram (view layer not shown)

3.3 Mobile implementation

3.3.1 Home

The home screen is the first thing that the user sees when opening the application.

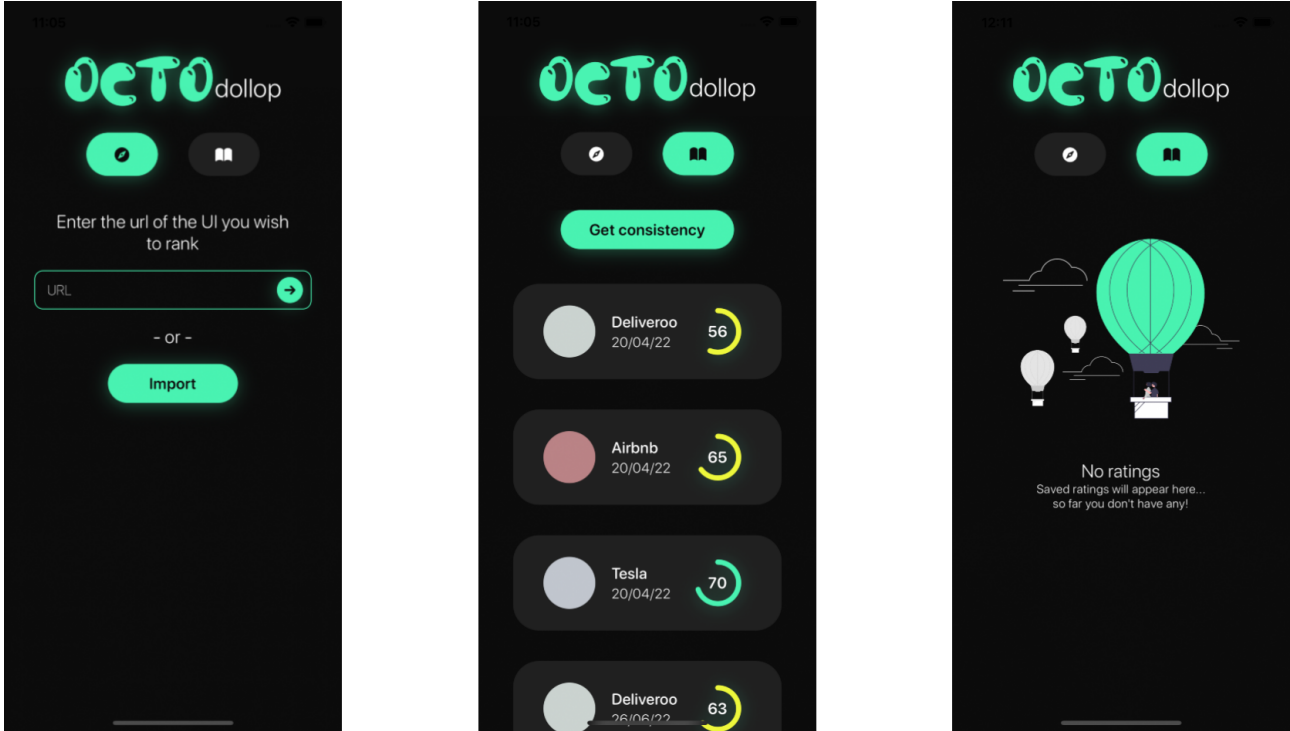


Figure 3.7: The home screen of Octodollop.

Left: ScreenInputViewController

Middle: HistoryViewController, saved UI ratings present

Right: HistoryViewController, no saved UI ratings

In the left-hand side screen (image 3.7), the user is able to input the UI he wishes to rate by either typing the url of a website in the highlighted search bar and then tapping on the green arrow, or by selecting the import button and choosing a previously captured screenshot from the media library.

In the screen in the middle, a list of the saved UI ratings is displayed. Each one of them is identified by a name, inputted by the user during the saving process, the date in which it was generated, the overall achieved score and the main theme color of the user interface. The theme color, which could be more accurately defined as the "average theme color", is calculated by leveraging the native iOS CoreImage framework. After extracting the corresponding `cilimage` from the target `UIImage` representation of the screenshot, the `CIAreaAverage` filter is applied to it, obtaining a single pixel that contains the average color for the entire image. The pixel is then converted into a 4-bit RGBA bitmap, whose values are used to instantiate the final average `UIColor`.

In the history screen, the user is also able to compute the consistency score between two or more saved UIs (more on this in 3.3.5) by tapping the corresponding button. In case the user doesn't have enough saved ratings, the "Get consistency" prompt will be hidden, impeding the metric computation procedure from ever beginning.

By tapping on a rating, all the additional information relative to it will be displayed (see 3.3.4).

The user is able to navigate between the input section and the history section by using the two buttons right below the Octodollop logo. By tapping on either one of them, the displayed `UIViewController` will be horizontally translated out of the screen's bounds while the new one is moved in.

The switch buttons, similarly to the application's logo, belong to `HomeViewController`, which also wraps `ScreenInputViewController` and `HistoryViewController` in order to make the screen transitions take place.

3.3.2 Input preview

The input preview screen is presented from `HomeViewController` whenever the user provides a UI as an input, either in the form of a URL or of a screenshot.

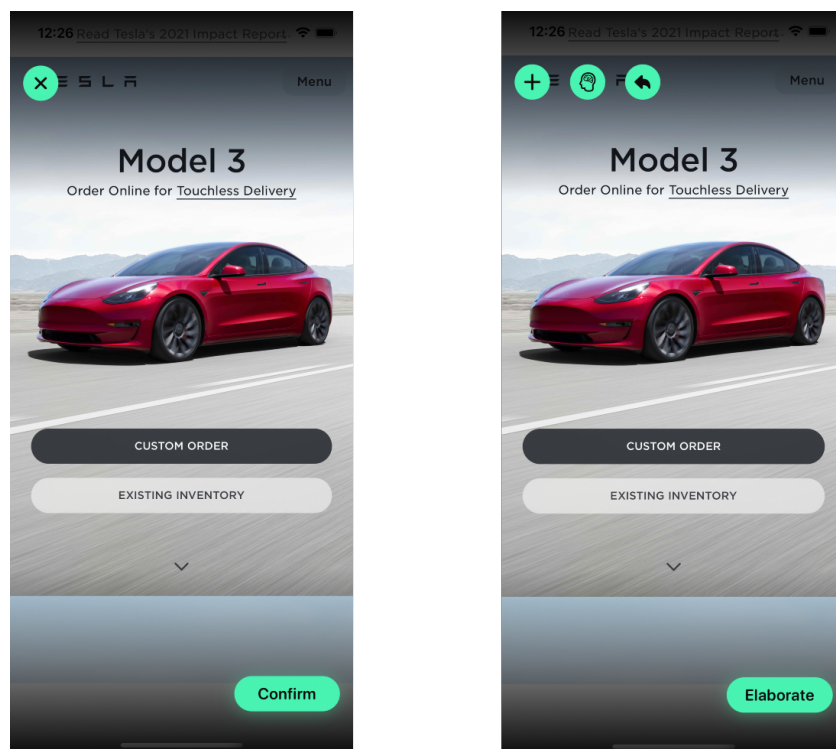


Figure 3.8: The input preview screen of Octodollop.

Left: `InputPreviewViewController`, when initialised with a URL

Right: `InputPreviewViewController`, when initialised with an image

In case a URL is a given, the user has the chance of browsing and navigating the related website, setting it up in whichever state he pleases. Once this setup phase is completed, he can then tap on the "Confirm" button in the bottom right corner to confirm his selection.

Behind the scenes, this will automatically trigger a screenshot capture of the web-page, whose `UIImage` representation is then presented and shown inside a `UIImageView`, all the while the buttons on the top left side of the screen become visible.

If a screenshot is provided instead, the screen on the right will be directly presented.

Through the button prompts on the top left corner, which appear on screen once `InputPreviewViewController` is provided with an image, the user is able to trigger all the input capture related actions.

By tapping on the "+" button, the manual input capture procedure (3.3.3) is triggered.

The "Head" button runs the artificial intelligence border recognition algorithm, outputting the automatically identified UI elements. The work related to this step is entirely offloaded to the server, so, client-wise, this translates to a simple network request being made to `/ai/bounding_boxes`. Clicking the "back arrow" button undoes the addition of any previously identified UI elements. This is done in blocks, meaning that if the user identified, for example, four elements in one input capture session and five elements in the next, tapping on the back button will remove five elements at first and, if clicked again, the remaining four elements. In case no identified elements are left, the button will trigger a dismissal of the rating process instead.

If the user selects the "Elaborate" button, visible on the bottom right of the screen, a `/rating` request will be made to the server and, after receiving the response with the final UI rating and its related aesthetics metrics, these results will be displayed (see 3.3.4).

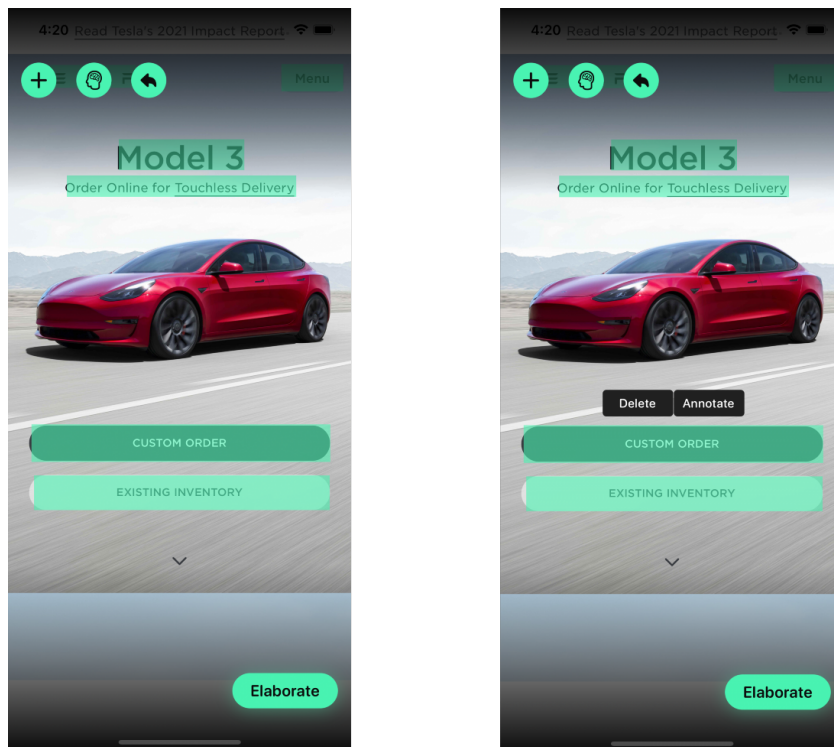


Figure 3.9: `InputPreviewViewController` displaying identified UI elements

In figure 3.9, user-identified UI elements are displayed as a green transparent overlay. By long-tapping any of the overlays, a contextual popup with the "Delete" and "Annotate" actions will appear right above it. While the delete action intuitively allows the removal of individual elements, the "Annotate" option enables the user to add a brief, explicative comment from a list of pre-made options. This additional functionality will be later on explored in 5.2.3.

3.3.3 Input gathering

Thanks to `InputGatheringViewController`, the user is able to manually identify UI elements, leveraging high precision interactions made possible by the native mobile application environment. Rectangular overlays are drawn by dragging one's finger across the screen, with the additional possibility of zooming-in the target UI and scrolling through it for added comfort and precision.

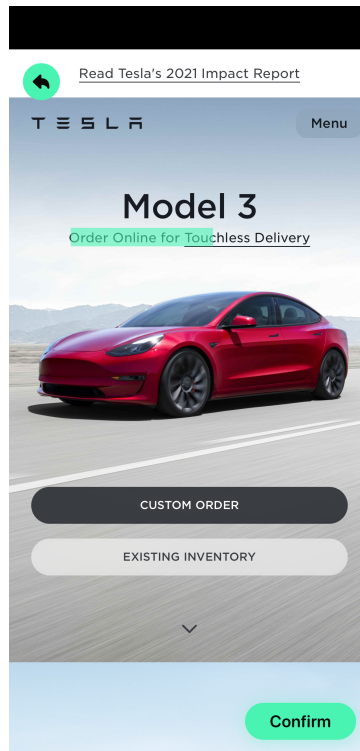


Figure 3.10: `InputGatheringViewController`, Octodollop’s input gathering screen, while an element is being identified

`InputGatheringViewController` is presented as nothing more than a full screen image, a back button on the top right corner and a confirm button on the bottom right. The back button undoes, one by one, the addition of any drawn overlays, while the confirm button saves them and pops back to `InputPreviewViewController`.

Both the back and the confirm buttons can be hidden by simply tapping on the screen, to avoid them getting in the way of the identification procedure.

Here below follows the pseudocode of the algorithm used for drawing overlays, where the `drawingChanged` function is to be intended as being triggered every time the user drags his finger across the screen and executed sequentially on a single thread.

```

1  startingPoint: Point := nil
2  drawOrigin: Point := nil
3  drawWidth: Float := nil
4  drawHeight: Float := nil
5
6  func doDrawing(gesture: GestureRecogniser) {
7      if gesture.state is .begun {
8          startingPoint = gesture.point
9      }
10     if gesture.state is .changed {
11         drawWidth = abs(gesture.point.x - startingPoint.x)
12         drawHeight = abs(gesture.point.y - startingPoint.y)
13         drawOriginX := min(gesture.point.x, startingPoint.x)
14         drawOriginY := min(gesture.point.y, startingPoint.y)
15         drawOrigin = Point(drawOriginX, drawOriginY)
16         draw(drawOrigin, drawWidth, drawHeight)

```

```

17     }
18     if gesture.state is .ended {
19         save(drawOrigin, drawWidth, drawHeight)
20         resetState()
21     }
22 }
23
24 func draw(origin: Point, width: Float, height: Float) {
25     // Draws on screen a rectangle with the given origin, width and height
26 }
27
28 func save(origin: Point, width: Float, height: Float) {
29     // Saves overlay to data source
30 }
31
32 func resetState() {
33     startingPoint = nil
34     drawOrigin = nil
35     drawWidth = nil
36     drawHeight = nil
37 }

```

3.3.4 UI rating

The UI rating screen is displayed in two separate occasions, either when the user receives the rating for the first time after requesting its computation in `InputPreviewViewController` (see 3.3.2), or by selecting a previously saved rating in `HistoryViewController` (see 3.3.1).

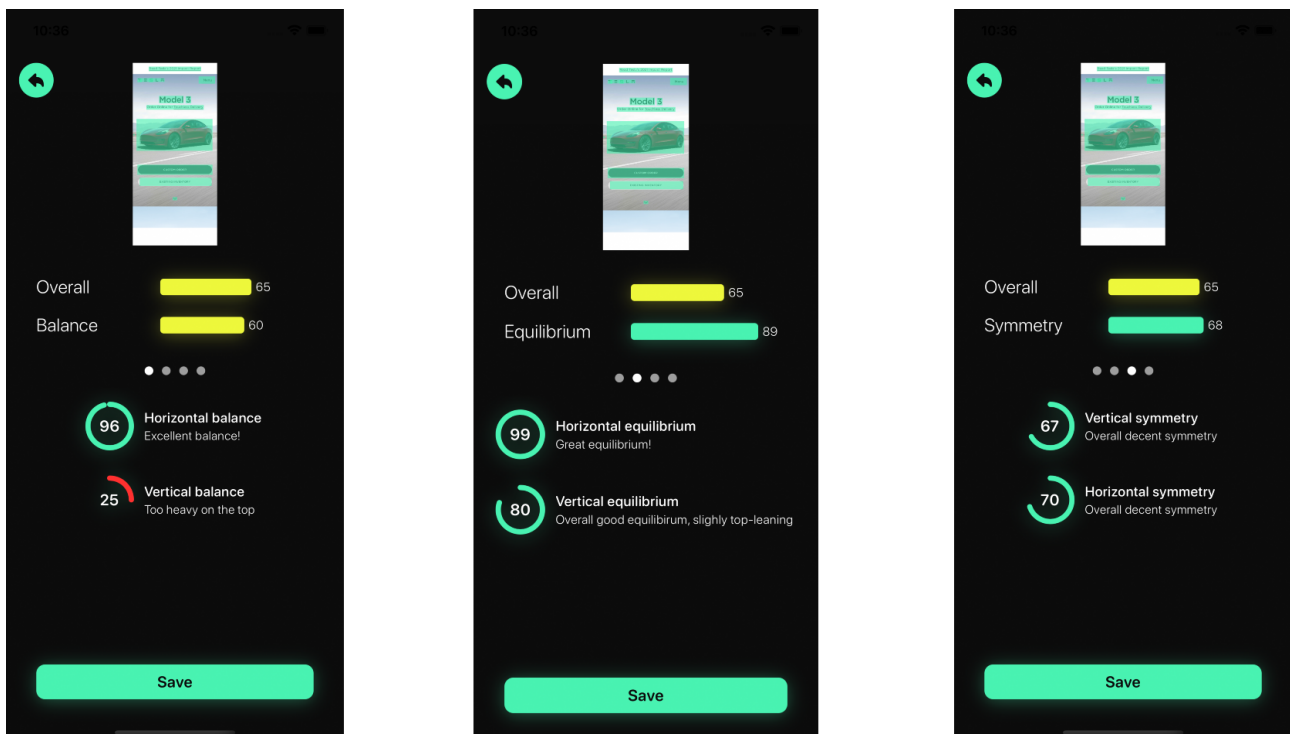


Figure 3.11: RatingViewController displaying a UI beauty score (only first three sections shown)

`RatingViewController` displays on the very top of the screen a preview of the rated UI, alongside its identified UI elements overlaid on top of it.

Right below, an overview of the metrics results is shown. While the "Overall" rating always stays the same and displays the UI average beauty score, the content of the bar underneath it can vary: by scrolling horizontally, the user can switch between different aesthetic metrics, grouped by section, and, by doing so, change both the displayed category rating and the related aesthetic metrics beneath it.

The presented metrics are nearly entirely server driven, meaning that integrating new ones, provided that the API endpoint returns them in the same standard format shown in [A.1.2](#), is as easy as adding new category identifiers to the `MetricType` enumeration.

The save button at the very bottom of the screen is displayed only if the rating hasn't been saved to the local database yet. Once tapped, the user will be prompted to enter a name for the UI and, consequently, the rating will be saved.

The saving procedure consists in, after saving the UI image for later access in the app's private documents directory, setting a name and creation date to the corresponding `Rating` model, followed by linking the saved image path to the model itself and, similarly, the related `UIElement` models, representing the user-identified UI elements. The average rating color, consisting in the longest step of the saving process and already detailed in [3.3.1](#), is then calculated and associated to the corresponding `Rating` model.

Finally, the updated model gets encoded to a raw JSON data object and written in the app's documents directory for later access.

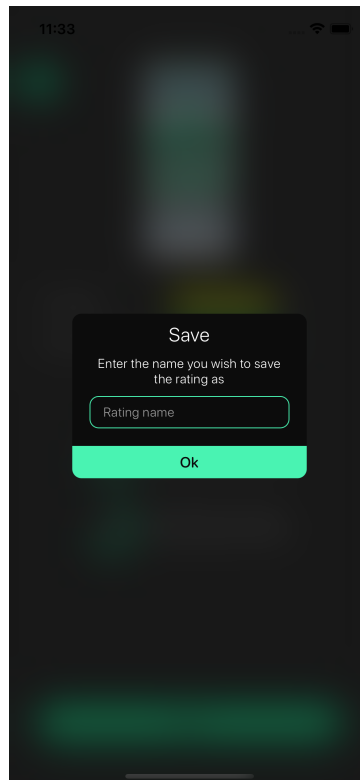


Figure 3.12: Popup in `RatingViewController` prompting the user to enter a name for the UI, shown once the save button is tapped

3.3.5 Consistency metric

The consistency metric is the only aesthetic metric in Octodollop which is entirely computed locally. The reason behind this choice was that, unlike the standard UI rating process which might need help from the server for image processing purposes, consistency scores only need data that is already available offline on the user's device: saved UI ratings.

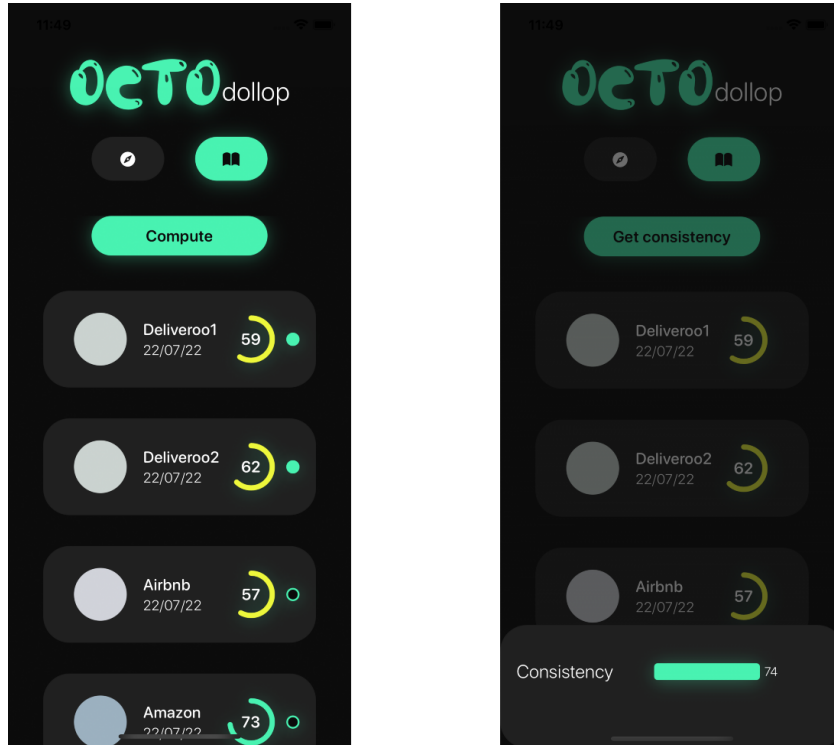


Figure 3.13: *Left*: HistoryViewController, selecting UIs to be fed into the consistency algorithm
Right: ConsistencyViewController, displaying the consistency score

By starting the consistency procedure in HistoryViewController (3.3.1), the user gets prompted to select two or more saved ratings. Selected ratings can be distinguished from non-selected ones based off the circle on their right-hand side being filled or not. Selections can be toggled by simply tapping on the target rating itself.

Clicking on the "Compute" button will trigger the presentation of ConsistencyViewController, which, given as input two or more Rating models, computes and displays their consistency. In short, consistency is meant to be calculated amongst different screens from the same app and represents how coherent they are between each other. Implementation details and a more in-depth explanation of the metric can be found at 3.4.6.

3.4 Aesthetic metrics implementation

All of the aesthetic metrics detailed here below, unless stated otherwise, are computed by the server after being given as input a series of identified UI elements, in the format already explained in 3.2.2 and shown in A.1.1.

The coordinate system used in each of the formulas below is to be considered as a 2D Cartesian system, in which the center is located in the middle of the phone's screen.

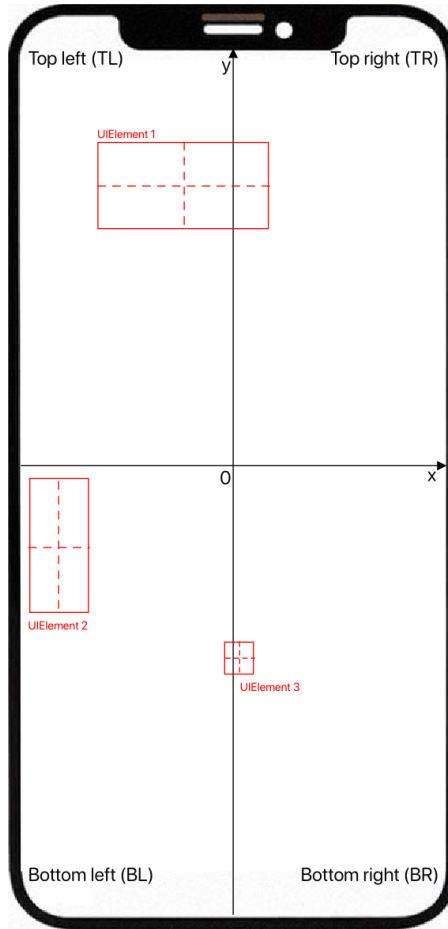


Figure 3.14: The coordinate system of a phone's screen

The left side of the screen is defined as the union of the TL and the BL quadrants, while the right side as the union of the TR and BR ones. Bottom and top sides are respectively identified by the union of the TL and TR quadrants and BL and BR ones.

3.4.1 Balance

Balance, distinguishable between vertical and horizontal balance, is responsible for giving a sense of stability and soundness to a user interface. It can be accomplished by maintaining an equal weight of the components on the opposite sides of the screen: either left and right when evaluating horizontal balance, or top and bottom when considering vertical balance instead.

A UI element is said to belong to either side of the screen, or to either quadrant, if its center falls within the bounds of the quadrant itself. In figure 3.14, for example, "UIElement 1" belongs to the Top Left quadrant, to the top and left side of the screen. An element whose center falls exactly on either one of the Cartesian axis, doesn't belong to either one of the relative opposite screen sides.

The weight w of a UI element is calculated as follows:

$$A = \text{width} \cdot \text{height}$$

$$w = A \cdot |\text{center}_x| \cdot |\text{center}_y|$$

An element whose center is exactly on one of the axis will have a weight of 0, resulting in it ultimately not being accounted for in the balance formula.

Defining w_L , w_R , w_T , w_B as the weight of the elements on the left, right, top and bottom side of the screen, each one of them calculated as the algebraic sum of the weight of the components belonging to it, the horizontal and vertical balance can be mathematically defined as follows:

$$w_x = \sum_i w_i$$

$$hb = \frac{w_L - w_R}{\max(w_L, w_R)}$$

$$vb = \frac{w_T - w_B}{\max(w_T, w_B)}$$

$$\text{horizontal balance score} = 1 - |hb|$$

$$\text{vertical balance score} = 1 - |vb|$$

Figure 3.15: The vertical and horizontal balance equations, adapted from Ngo et al. [12]

Low balance scores will be closer to 0, while user interface with good balance will be closer to 1. Alongside each score, the API returns a brief comment explaining it, concisely described by the pseudocode here below, in which the variables **hb** and **vb** refer to the values obtained through 3.15.

```

1  if axis is .horizontal. and hb < 0 {
2      direction := "right"
3  } else if axis is .horizontal and hb >= 0 {
4      direction := "left"
5  } else if axis is .vertical and vb < 0 {
6      direction := "top"
7  } else if axis is .vertical and vb >= 0 {
8      direction := "bottom"
9  }
10 if score <= 0.3 {
11     message := "Too heavy on the " + direction
12 } else if score <= 0.5 {
13     message := "Heavy on the " + direction
14 } else if score <= 0.7 {
15     message := "Slightly heavy on the " + direction
16 } else if score <= 0.8 {
17     message := "Good balance, slightly " + direction + "-heavy"
18 } else {
19     message := "Excellent balance!"
20 }

```

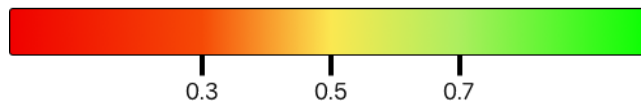


Figure 3.16: Indicative balance scores quality meter

3.4.2 Density

Density relates the percentage of the screen occupied by content with the remaining space that is instead left empty. When designing user interfaces, it is good practice to keep a balance between filling too much and cluttering a screen with elements versus an excessive and overzealous minimalism, resulting in near empty and gaunt screens.

By defining $width_c$ and $height_c$ as the width and height of the phone's screen, the density metric can be computed from the contribution of the individual screen's components.

$$d = \frac{\sum_i width_i \cdot height_i}{width_c \cdot height_c}$$
$$\text{Density score} = 1 - \frac{|d - 0.65|}{0.65}$$

Figure 3.17: The density equation, adapted from Zen [17]

In formula 3.17, 0.65 is a constant experimentally tuned factor introduced for better ratings accuracy. With d being the partial result from the just-mentioned formula, the metric results can be interpreted as follows.

```
1  if 0 <= d <= 0.35 {  
2      message := "The screen doesn't have enough content"  
3  } else if 0.35 < score <= 0.70 {  
4      message := "The screen space is well used"  
5  } else {  
6      "The screen appears too cluttered"  
7  }
```



Figure 3.18: Indicative density scores quality meter

3.4.3 Equilibrium

Equilibrium can be defined as the balance between opposing UI elements, which, in this context, can be seen as opposing forces. Similarly to balance, equilibrium is divided into its horizontal and vertical components; a layout is said to be in equilibrium the closer that its center's x_l and y_l coordinates are to the canvas' origin (x_c, y_c) . The center of a layout can be calculated as the sum of the individual contributions of each component, as per the following formula.

$$x_l = \frac{\sum_i \text{width}_i \cdot \text{height}_i \cdot \text{center}_{x_i}}{\sum_i \text{width}_i \cdot \text{height}_i}$$

$$y_l = \frac{\sum_i \text{width}_i \cdot \text{height}_i \cdot \text{center}_{y_i}}{\sum_i \text{width}_i \cdot \text{height}_i}$$

$$he = \frac{x_c - x_l}{\max(x_c, x_l)}$$

$$ve = \frac{y_c - y_l}{\max(y_c, y_l)}$$

$$\text{horizontal equilibrium score} = 1 - |he|$$

$$\text{vertical equilibrium score} = 1 - |ve|$$

Figure 3.19: The vertical and horizontal equilibrium equations, adapted from Ngo et al. [12]

Qualitatively, the equilibrium scores have been interpreted with similar thresholds to the balance metric, where **he** and **ve** refer to the partial results obtained with formula 3.19.

```

1  if axis is .horizontal and he < 0 {
2      direction := "right"
3  } else if axis is .horizontal and he >= 0 {
4      direction := "left"
5  } else if axis is .vertical and ve < 0 {
6      direction := "bottom"
7  } else if axis is .vertical and ve >= 0 {
8      direction := "top"
9  }
10 if score <= 0.3 {
11     message := "Heavy disequilibrium towards the " + direction
12 } else if score <= 0.5 {
13     message := "Visible disequilibrium towards the " + direction
14 } else if score <= 0.7 {
15     message := "Slight disequilibrium towards the " + direction
16 } else if score <= 0.8 {
17     message := "Overall good equilibrium, slightly" + direction + "-leaning"
18 } else {
19     message := "Great equilibrium!"
20 }

```

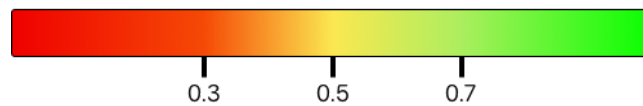


Figure 3.20: Indicative equilibrium scores quality meter

3.4.4 Proportion

Proportion can be described as the similarity between the shapes of the elements in a user interface. As in our model's abstraction all the identified UI elements are effectively rectangles, shape similarity can be simplified to rectangles aspect ratio likeness.

Defining r as the aspect ratio of each component, proportion is then found from the contribution of the individual elements.

$$p = \sum_i \sum_j \frac{|r_i - r_j|}{\max(r_i, r_j)}, i \neq j$$
$$\text{proportion score} = 1 - \frac{p}{i}$$

Figure 3.21: The proportion equations

The score's meaning is explained as per the pseudocode below, where p is the partial result obtained through formula 3.21.

```
1  if p <= 0.3 {  
2      message := "The shape of your elements differs greatly"  
3  } else if p <= 0.5 {  
4      message := "The elements are not entirely homogeneous"  
5  } else if p <= 0.7 {  
6      message := "The elements are consistently shaped"  
7  } else {  
8      message := "The elements proportions are excellent!"  
9  }
```

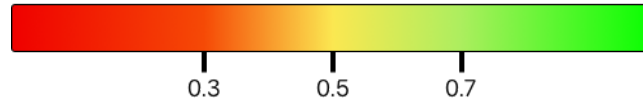


Figure 3.22: Indicative proportion scores quality meter

3.4.5 Symmetry

Symmetry represents the harmonious displaying of elements around an axis (symmetry axis). In a way, symmetry can be seen as a form of duplication, as in perfectly symmetrical UIs, a shape present on one side of the axis will have an exact specular correspondence on the other side.

Mathematically, defining x_c and y_c as the origin of the phone's screen and TL, TR, BL, BR as the four quadrants of our coordinate system (fig. 3.14), the horizontal and vertical components of symmetry can be calculated by taking into account each component in the user interface, as detailed in the following formulas.

$$g_x = \sum_i (|x_i - x_c|, |y_i - y_c|, \text{width}_i, \text{height}_i)$$

$$s_{\text{top}} = \frac{g_{TL} - g_{TR}}{\text{maxvect}(g_{TL}, g_{TR})}$$

$$s_{\text{bottom}} = \frac{g_{BL} - g_{BR}}{\text{maxvect}(g_{BL}, g_{BR})}$$

$$s_{\text{left}} = \frac{g_{TL} - g_{BL}}{\text{maxvect}(g_{TL}, g_{BL})}$$

$$s_{\text{right}} = \frac{g_{TR} - g_{BR}}{\text{maxvect}(g_{TR}, g_{BR})}$$

Where *maxvect* is a special operation which, given as input two vectors in $\mathcal{R}^n$, returns a vector in $\mathcal{R}^n$ where each component is the maximum between the two input vectors. E.g:

$$\text{maxvect}((1, 2, 3), (0, 3, 4)) = (1, 3, 4)$$

Remembering that each one of the *s* vectors is defined in the $\mathcal{R}^4$ vectorspace, the final

symmetry scores are then

$$\text{vertical symmetry score} = \frac{\frac{|\sum_i^4 s_{\text{top}_i}|}{4} - \frac{|\sum_i^4 s_{\text{bottom}_i}|}{4}}{2}$$

$$\text{horizontal symmetry score} = \frac{\frac{|\sum_i^4 s_{\text{left}_i}|}{4} - \frac{|\sum_i^4 s_{\text{right}_i}|}{4}}{2}$$

Figure 3.23: The vertical and horizontal symmetry equations, adapted from Ngo et al. [12]

The final scores are interpreted as such:

```

1  if score <= 0.3 {
2      message := "Significant asymmetry in the layout"
3  } else if score <= 0.5 {
4      message := "Noticeable asymmetry in the layout"
5  } else if score <= 0.7 {
6      message := "Overall decent symmetry"
7  } else if score <= 0.8 {
8      message := "Very good symmetry"
9  } else {
10     message := "Superb symmetry!"
11 }

```

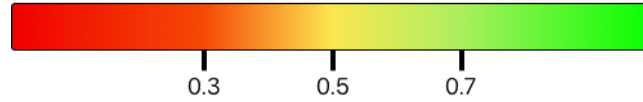


Figure 3.24: Indicative symmetry scores quality meter

3.4.6 Consistency

Only metric amongst the one mentioned so far being computed locally on the mobile client, an application can be defined as consistent if its elements across different screens (buttons, labels, images, ...) are similar in position, dimension, arrangement and shape.

It has been calculated as a straightforward implementation of the formula elaborated by Burny et al. in his research [3].

3.4.7 Future work

In order to aid the development process of any metrics that might get possibly integrated in the future, it will now be described how making any such additions can easily be achieved.

1. Choose the metric to be implemented and its corresponding formula.
2. Server-side, create a class `MyMetricRater` as a concrete implementation of the `Rater` abstract class. The rater will need to be able to process `Canvas` and `Element` models in the same coordinate system that has been described thus far.
3. Upon completion of all the computations in the `MyMetricRater`'s `rate` method, return a list containing the results in the form of `Rating` models. Each `Rating` model will need to be initialised with a unique identifier, a rating from 0 to 100 and a human readable comment explaining the achieved score.
4. In `rater/__init__.py`, add `MyMetricRater` to the list of available raters.
5. In `rating/views.py`, add `MyMetricRater` to the list of raters to be used when computing the final result.
6. In the iOS app, add the ids of the `Ratings` returned by `MyMetricRater` to the `MetricType` enumeration, alongside their human readable description.

3.5 Borders recognition model implementation

In order to aid the manual UI element identification procedure, an OpenCV-powered border recognition model has been developed. Given as input a screenshot of a user interface, the model will identify all the UI components in it, converting them into a model usable by the rest of the application.

The various versions of the model will now be described and analyzed, highlighting their respective strengths and shortcomings. Output examples, running the various algorithms over the Deliveroo and Nodalview apps homepage, will additionally be shown for each model revision.

3.5.1 First attempts

Originally, attempts of processing the images offline directly on the users phone were made, with the intention of leveraging Apple's native Vision framework and its image processing APIs, allowing for quick (<1 second) computations without the need of making network requests.

Ultimately, the `VNDetectContoursRequest` object had been used, but the results weren't satisfactory. Even by tuning its parameters to increase the input image contrast for better

border recognition performance, the results were very poor. The detected elements contained a lot of noise (false positives), especially around colorful images, in which different objects were marked as distinct elements. An attempt at filtering out all the unnecessarily detected object has been made by removing those whose width and height didn't meet certain thresholds, but even with these manual tweaks the overall quality was simply not high enough: lots of elements were left unrecognised and the noise, although reduced, was still significant.

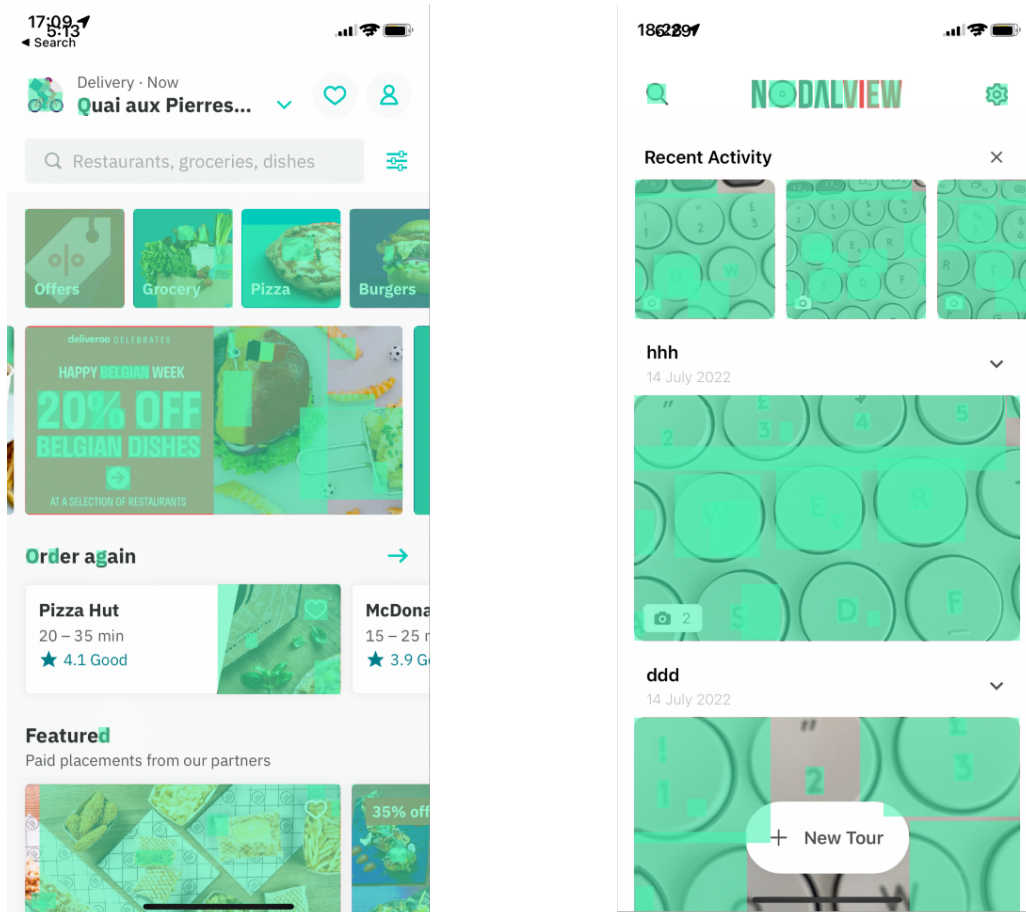


Figure 3.25: The results obtained with the final version of the iOS Vision-framework-driven border recognition model

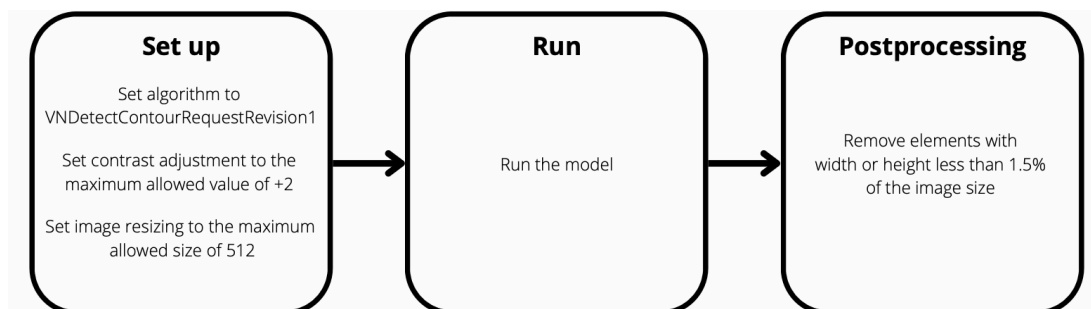


Figure 3.26: The final algorithm used to obtain the results in fig. 3.25

3.5.2 OpenCV - first version

Following the unfruitful attempts with the Vision framework, the approach was changed completely and it was decided to offload the entire border recognition work to the server, in order to use the powerful image processing library OpenCV.

While other implementation options have been researched for this purpose, none of them was considered fitting for the project, as the most popular vision-oriented frameworks are either OpenCV wrappers/extension themselves or more geared towards machine learning and data science tasks (TensorFlow, Scikit-Image, ...).

The approach was quite straightforward: convert the input image to grayscale and then simply run the `OpenCV.findContours` method to detect any UI elements. The results were immediately excellent, with minimal false positives detected and with great accuracy over the recognised elements, it was apparent that OpenCV was the way to go.

A visible defect was that, when it came to labels, each individual letter was detected as a separate object. While this behaviour isn't inherently undesired, it was decided that, for our objectives, it was better to mark an entire label as a single entity, meaning that further adjustments to model were to be made.

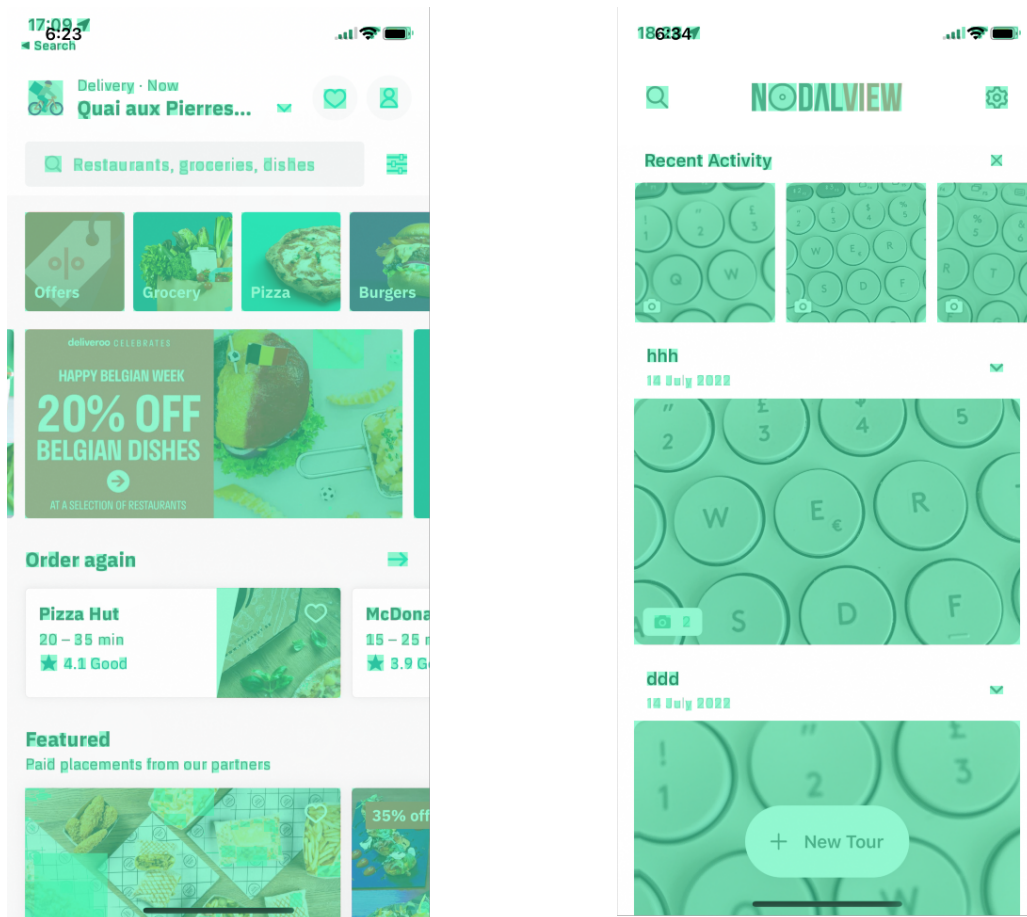


Figure 3.27: The results obtained with the first version of the OpenCV model

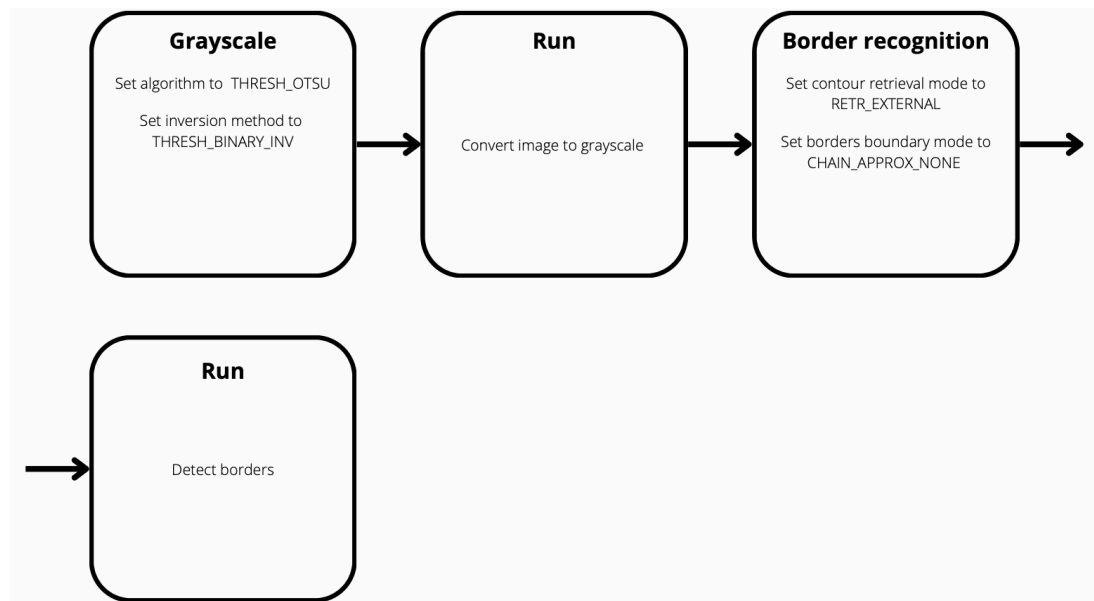


Figure 3.28: The various steps of the utilised algorithm

3.5.3 OpenCV - second version

As the second iteration of the model previously described, noise reduction and letters aggregation were posed as the main objectives. To get around those problems, the input image was initially blurred, following the intuition that, as the letters borders might this way become less well defined, with endings of one letter overlapping with beginnings of the next one, they would get detected as a single object altogether.

This idea proved to be sound, but larger labels, in which the spacing between each character is on average wider, were still not recognised as a single entity. For this reason, it was decided to first draw on the grayscale image the recognised borders, blur the result and then, finally, do a last pass of the border recognition step. This technique proved itself to be very effective, with letters no longer being marked as individual objects.

As a side effect, the noise got entirely removed from the final output, as the round of blurring made less relevant details, which were ultimately the cause for false positives, blend in with the background of their parent components, causing them not to be detected anymore.

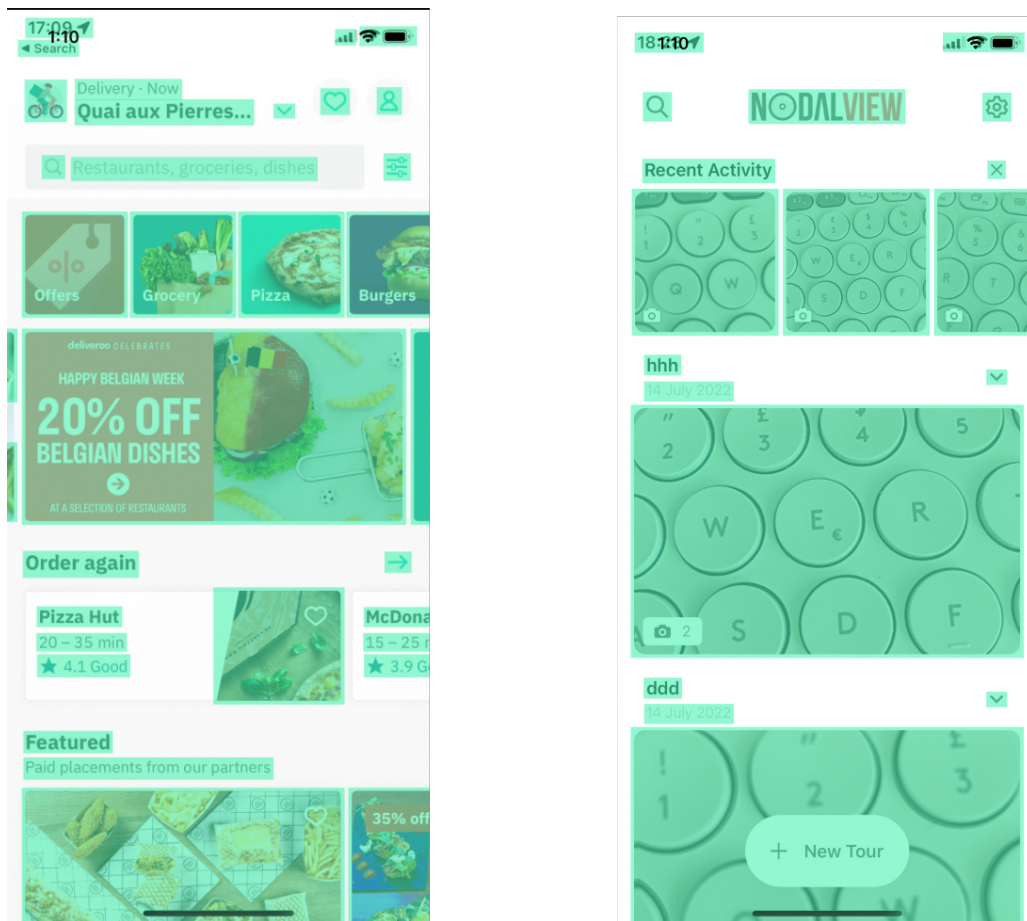


Figure 3.29: The results obtained with the second version of the OpenCV model

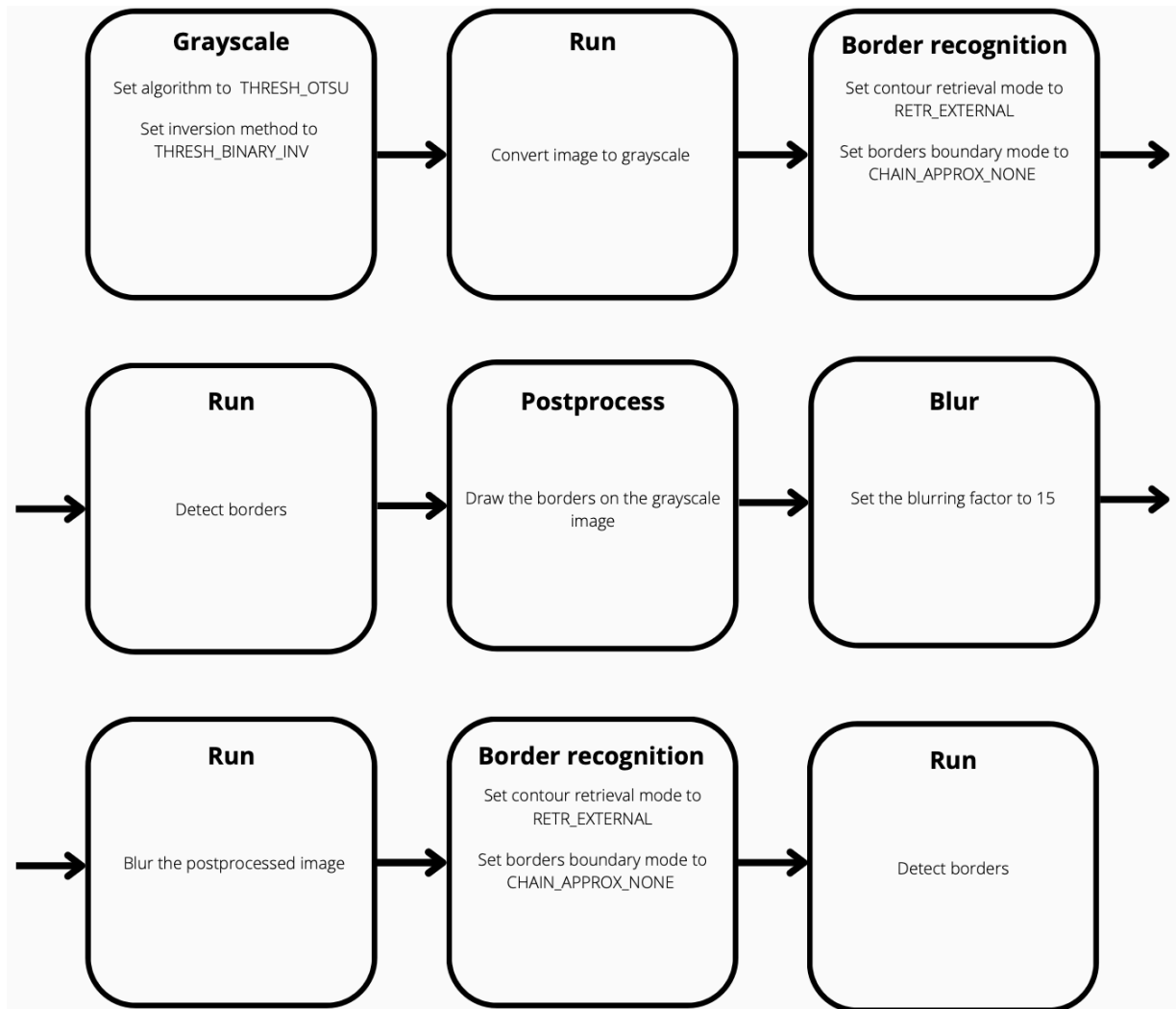


Figure 3.30: The algorithm of the second version of the OpenCV model

3.5.4 OpenCV - final version

After extensive testing, while the second version of the model worked extremely well under normal conditions, it was found that it failed to detect any elements at all when the target user interface had a dark background ("dark mode"), marking, in this case, only a single object enclosing the entirety of the canvas.

The root cause of this was pinpointed to the inversion method in the grayscale step of the algorithm: by hardcoding it to be `THRESH_BINARY_INV`, the background was always assumed to be made out of light colors, while for dark user interfaces it's necessary to use `THRESH_BINARY` instead.

In order to detect the main theme of the background and adjust the calculations accordingly, the threshold value found during the grayscale conversion is compared against 128, the midpoint of the grayscale itself, which ranges from 0 to 255. If the value is found to be less than the midpoint, it means that we are dealing with a UI in "dark mode", otherwise its a standard screen with a light background.

The rest of the algorithm is left unchanged.

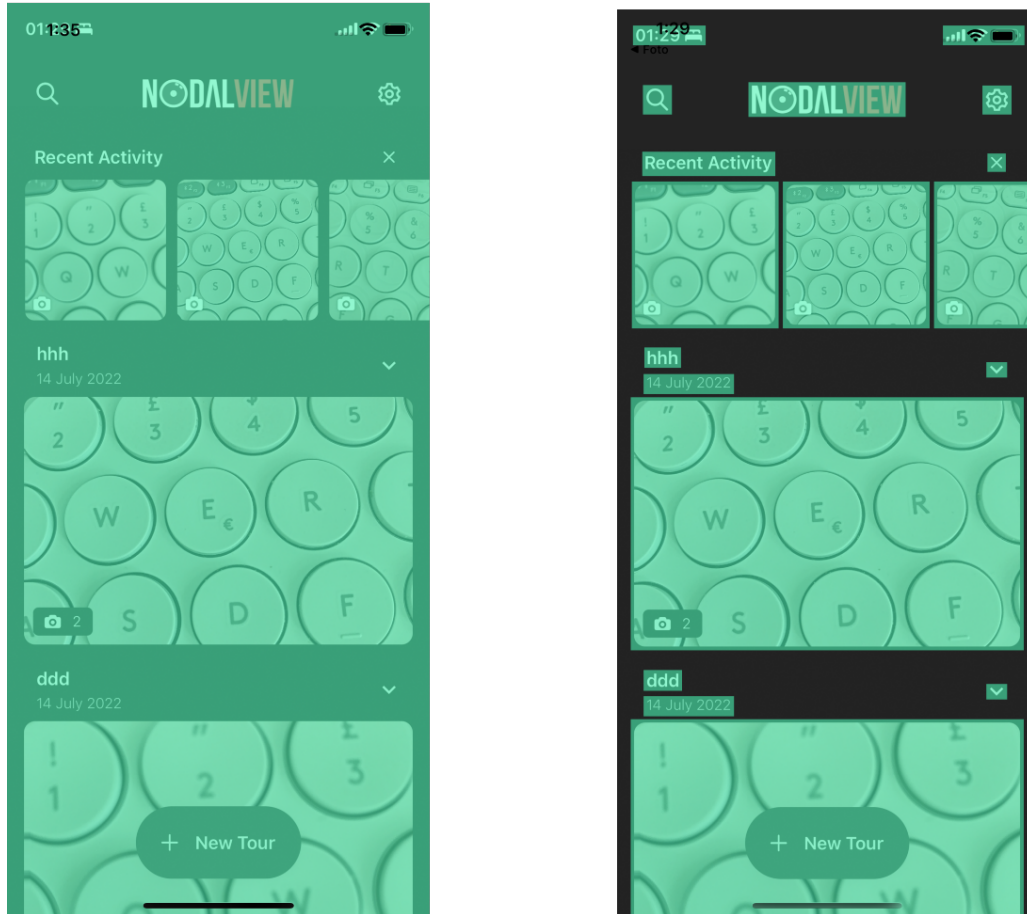


Figure 3.31: Comparison of results before grayscale threshold adjustments (left) and after (right) on a user interface in "dark mode"

3.6 Testing

On top of the acceptance testing that was continuously performed as more application features were developed, formal software testing methodologies and tools were used as well.

In regards to the iOS application, the suite XCTest was employed to add test coverage to the entirety of the business logic, consisting of the various screen's ViewModels.

It was purposely decided not to integrate any UI tests as, at least when it comes to iOS development, on top of the incredibly high cost associated to both running and maintaining them, they tend to be inherently flaky and unreliable. For this reason, manual quality assurance tasks, both on simulators and real devices, were carried out to ensure the usability and conformance of the app's UI to the initial planning.

Server-wise, on the other hand, each individual rater was unit-tested as to ensure its reliability when working with values outside, inside and at the edges of the operational range, on top of verifying that the outputted result was in the correct, expected format.

The border recognition model was instead manually tested on a large plethora of user interfaces and evaluated based on its balanced accuracy, taking into account both false positives (elements noise) and false negatives (unrecognised elements).

3.7 Download

The OctoDollop app can be downloaded from the Apple's App Store by simply searching for

"octodollop" or, alternatively, from the following link: [download](#).

Chapter 4

Software validation by examples

In this chapter, we will now use the developed application to rate the user interfaces of two iOS apps: Amazon and SNCB. For each one of them, two screens will be analysed in order to later on compare their overall consistency as well.

4.1 Amazon

4.1.1 Home

- **Horizontal balance:** the high score can be explained by the specular distribution of elements' weights, mostly consisting of horizontally scrollable collection views in which each object has the same sizing and inter-item spacing. (*score: 80*)
- **Vertical balance:** the very high score, which at first might seem counter-intuitive due to a big chunk of the user-interactable content being on the lower half of the screen, can be explained by the big camera image being positioned on the upper portion of the canvas, balancing the weights of the bottom-most collection views. On top of this, the tab bar is quite lightweight, consisting only of four small icons, further balancing out the vertical weight of the layout. (*score: 93*)
- **Horizontal equilibrium:** the very high score indicates an excellent soundness of the UI. The horizontally scrollable collection views, the full-width search bar and address label contribute to making the horizontal center of the layout very close to the actual origin of the phone's screen. (*score: 96*)
- **Vertical equilibrium:** the very high score is due to the heaviest items in the layout, consisting of the camera image, the "Keep shopping for" card, the "Pick up where you left off card" and the "Today's Deals" section, being very close, vertically speaking, to the origin of the screen. (*score: 98*)
- **Horizontal symmetry:** the very poor score can be traced back to the camera image and "Capturing moments" title and subtitle, which, although quite balanced and in equilibrium, are not horizontally symmetric. Similarly, the address label and the "Today's Deals" label are only present on the left side of the screen, with no equivalent right-hand-side component. (*score: 20*)
- **Vertical symmetry:** the very poor score can be motivated by the fundamentally and substantially different kind of elements presents in the top versus bottom part of the screen. (*score: 23*)
- **Density:** the very high score is explained by a smart use of the space. The UI isn't too cluttered, with small and empty gaps being filled with lightweight, non-bulky labels, while

- bigger ones are instead occupied by large images. (*score: 96*)
- **Proportion:** the medium score is due to a wide range of distinct type of elements of different sizes being employed throughout the UI. (*score: 41*)

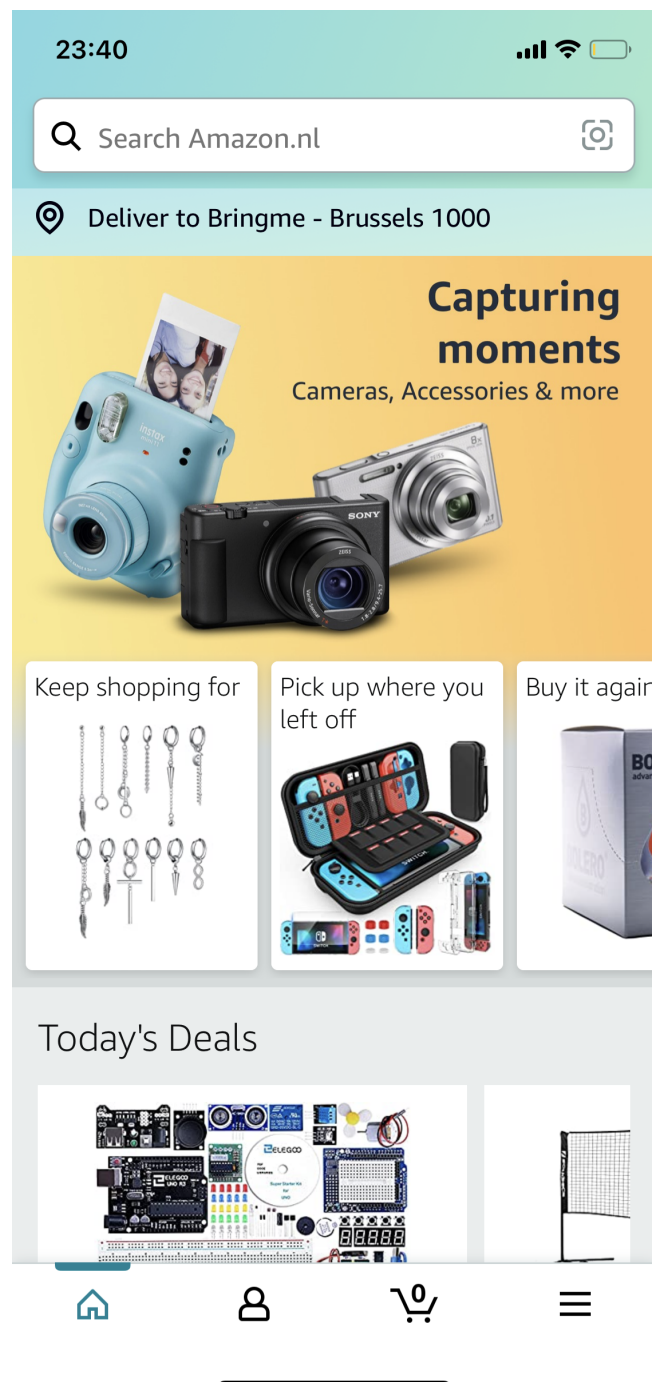


Figure 4.1: Amazon home screen

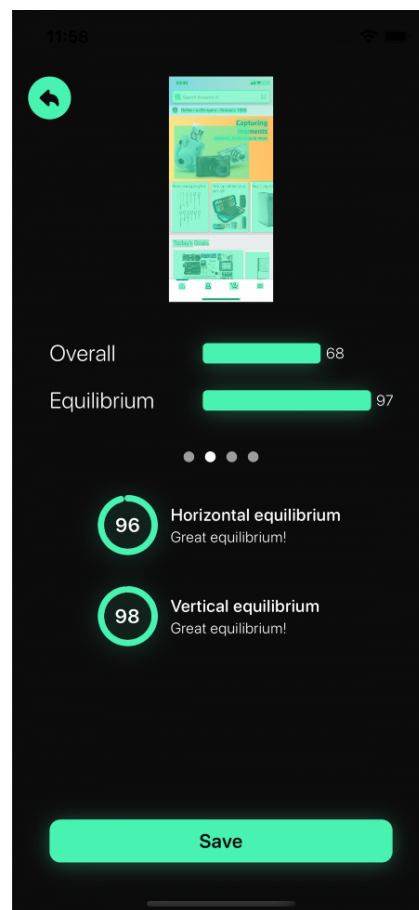
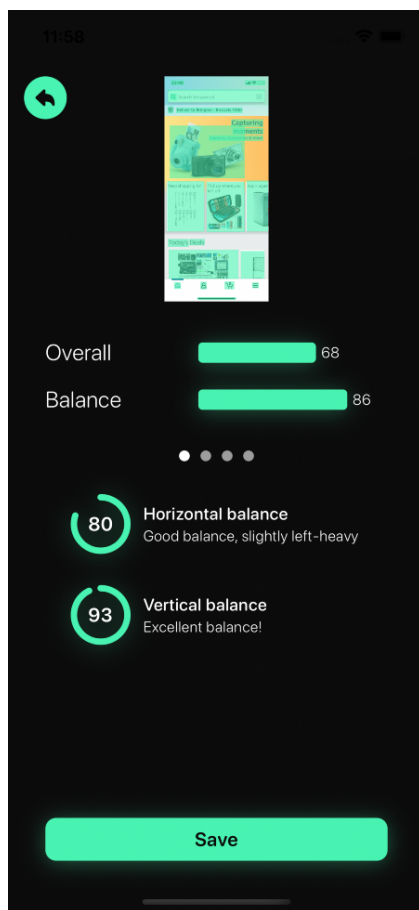


Figure 4.2: Amazon home balance and equilibrium ratings

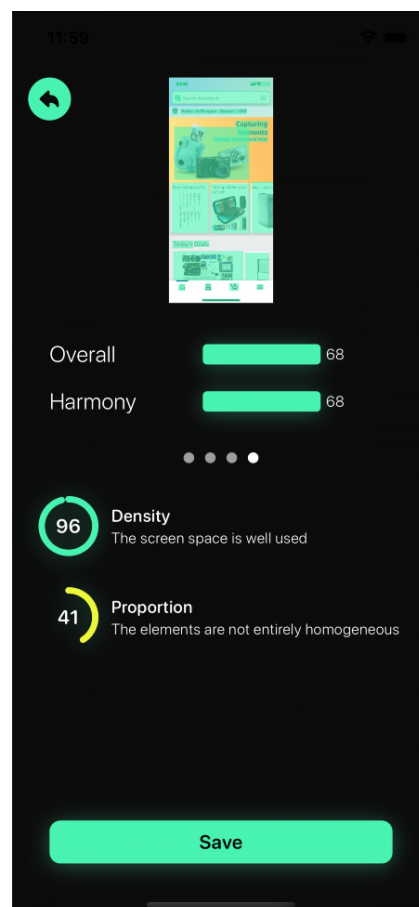
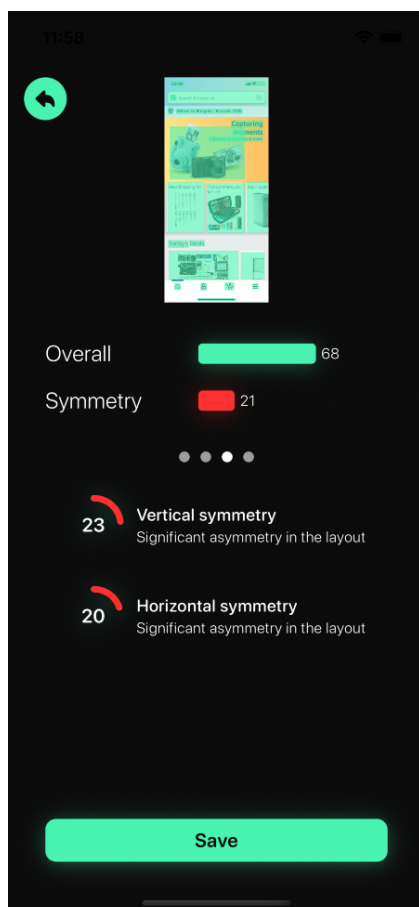


Figure 4.3: Amazon home symmetry and harmony ratings

4.1.2 Cart

- **Horizontal balance:** the medium score can be explained by the cart image on the top part of the screen. Due to it being very close to the left canvas margin and being counterbalanced only by a small lightweight label, the overall horizontal balance score, which would otherwise be relatively high, is brought down. (*score: 53*)
- **Vertical balance:** the medium to high score is achieved thanks to a generally well vertically balanced user interface. However, due to the elements in the "Recommendations for all products" section not having their own enclosing component, unlike the elements in the home screen (fig 4.1), the layout results a little unbalanced on its lower half. (*score: 66*)
- **Horizontal equilibrium:** the very high score is thanks to all the elements being close to canvas' center. (*score: 88*)
- **Vertical equilibrium:** the very high score is due to the biggest elements in the layout, consisting of the entire "Recommendations for all products" section, being very close to the screen's vertical origin. The few outliers, such as the tab bar, search bar and empty cart icon, are not big enough to significantly negatively impact the vertical equilibrium rating. (*score: 90*)
- **Horizontal symmetry:** the very poor score is mainly due to the majority of the UIs element being labels, with which symmetry is harder to achieve. The cart image on the top-left part of the screen, not matched by a similar element on the top-right side, is an additional cause of horizontal asymmetry. (*score: 19*)
- **Vertical symmetry:** the reasons behind the poor score are similar to the ones already detailed for the horizontal symmetry, with the additional factor of element in the lower part of the screen being significantly different than the ones on the top part. (*score: 21*)
- **Density:** the medium score can be explained by an excessive emptiness of the screen. The elements in "Recommendations for all products", by not being enclosed in their own distinct boxes, contribute to the UI not looking as content-dense as it could be. (*score: 54*)
- **Proportion:** the medium score is due to the user interface mainly consisting of labels, which all have different widths and heights. (*score: 45*)

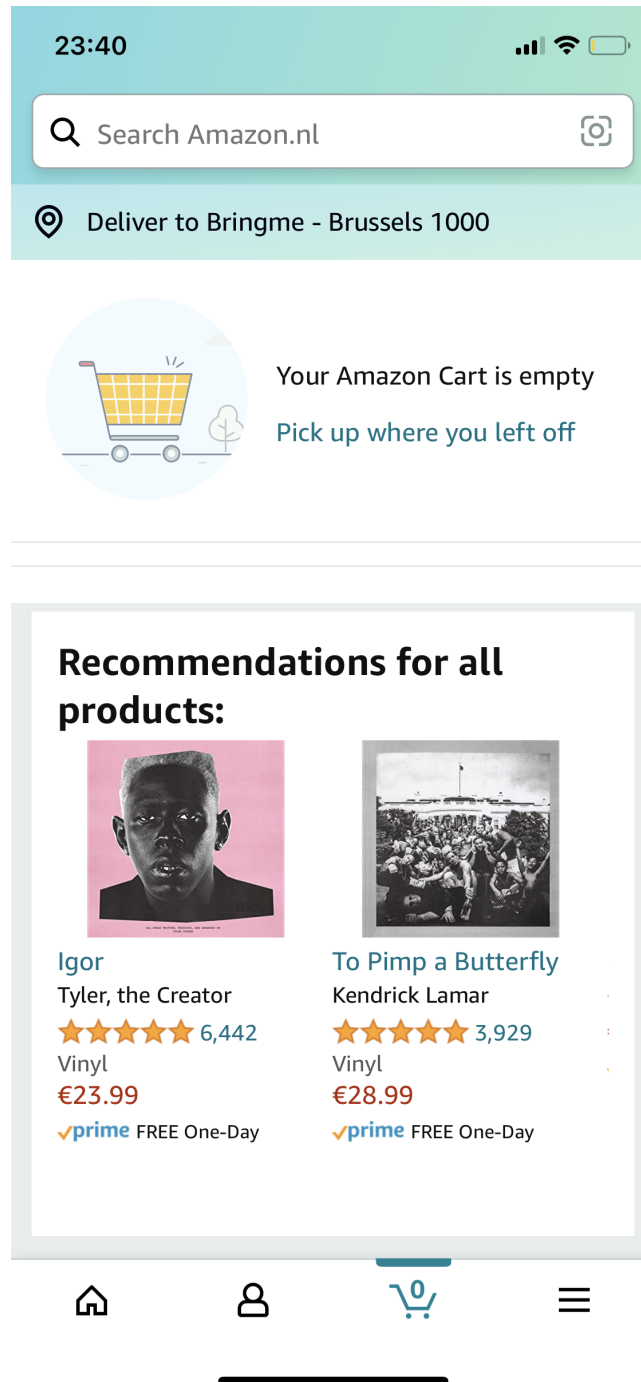


Figure 4.4: Amazon cart screen

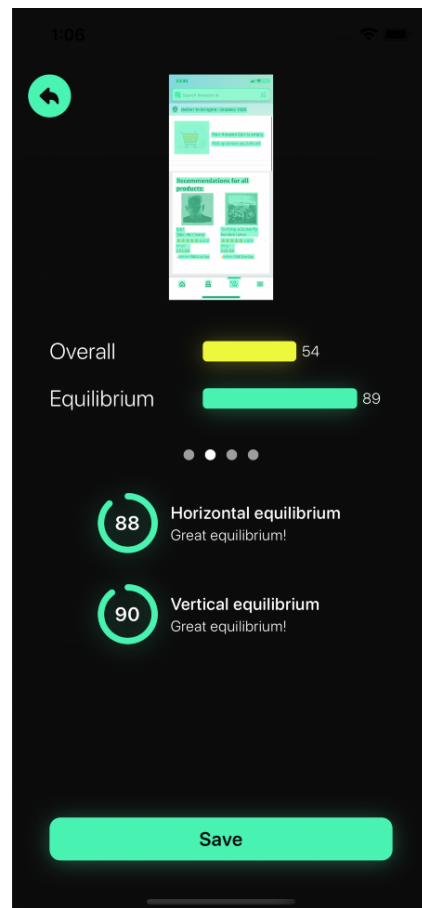
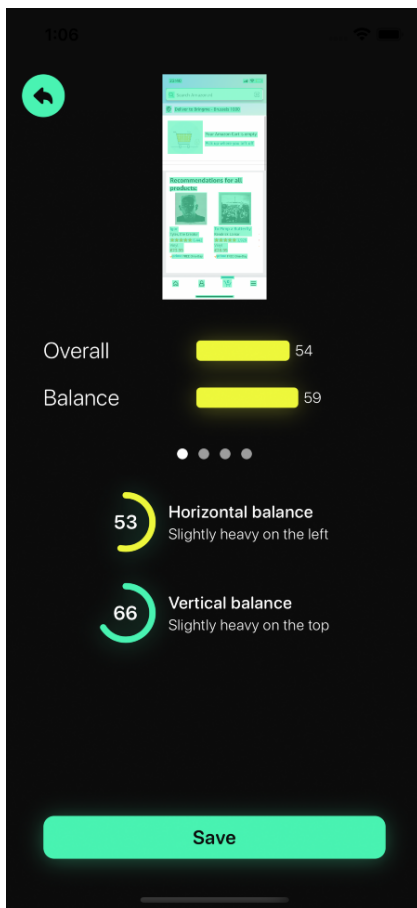


Figure 4.5: Amazon cart balance and equilibrium ratings

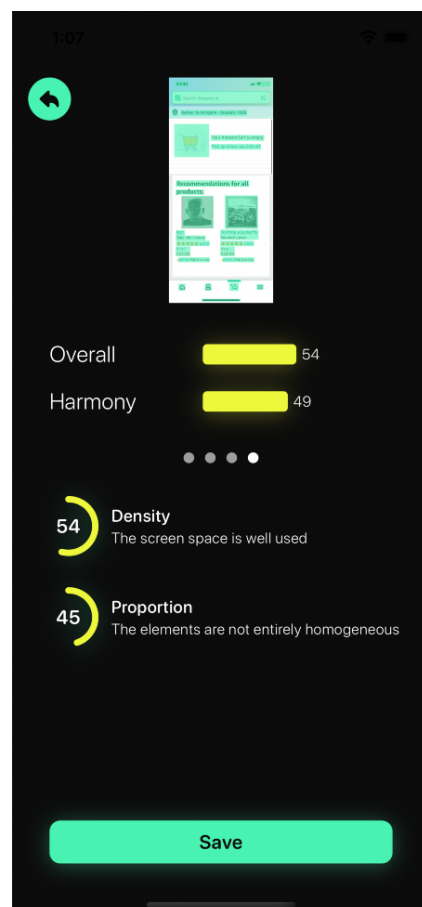
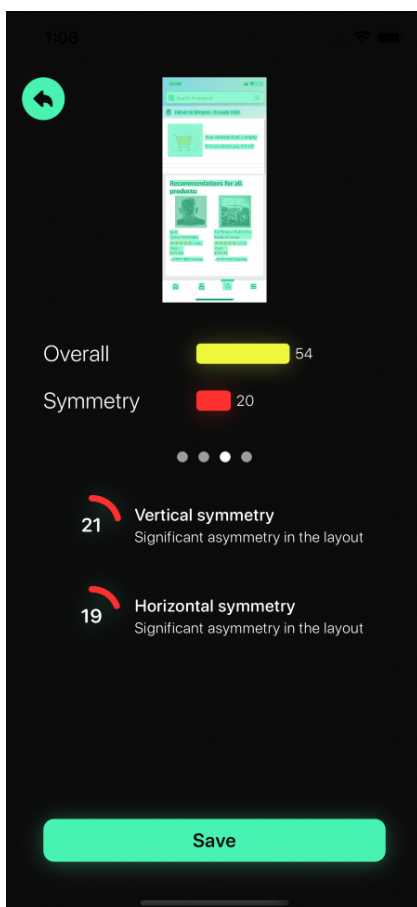


Figure 4.6: Amazon cart symmetry and harmony ratings

4.1.3 Consistency

The home and cart screen achieve a consistency rating of 87. This very high score is thanks to the same UI building blocks being used across different screen and them being placed in similar positions.

User content is concentrated in the bottom part of the screens, with each individual object having approximately the same size across the two UIs. The tab bar is exactly the same on both screens, and so is the navigation bar, made out of the search bar and address label. Additionally, both layouts feature an image on the top part of the canvas, counterbalanced by two labels on its very right.

4.2 SNCB

4.2.1 Home

- **Horizontal balance:** the high score is reconcilable with the fact that the near entirety of the elements are centered and spread horizontally across the whole screen. The ones that aren't, like for example the tab bar, are comprised out of identical, equally spaced elements. (*score: 81*)
- **Vertical balance:** the medium score is because most of the heavy components (images, text fields, icons) are on the upper half of the screen, although they are slightly balanced by the "Shortcuts" section. (*score: 40*)
- **Horizontal equilibrium:** similar to the horizontal balance, the very high score is justifiable by the fact that nearly all components are horizontally centered and spread over the entire screen, making their horizontal center very close to the canvas' origin. (*score: 97*)
- **Vertical equilibrium:** the high score is due to most of the UI objects being concentrated around the vertical center of the screen, with the exception of the big banner at the very top of the screen, which is however counterbalanced by the tab bar at its very bottom. (*score: 79*)
- **Horizontal symmetry:** the poor score is explained by the input text fields, which are entirely left leaning with no right counterpart. Similarly, the "arrow" icon element is only present on the right side of the screen, with nothing balancing it on the left side. The rating is slightly brought up by the rest of the full-width elements and by the tab bar. (*score: 35*)
- **Vertical symmetry:** the very poor score is motivated by the elements on the top half of the screen not having any specular match at all on the bottom half. (*score: 19*)
- **Density:** the poor score can be motivated by the obvious lack of content in the screen, with huge blocks of empty space, such as the one right below the "Shortcuts" section, being completely unused. (*score: 38*)
- **Proportion:** the medium score is due to the small number of elements being present on screen. The few objects visible in the UI are in fact often similar, or even near identical, in pairs. (*score: 42*)

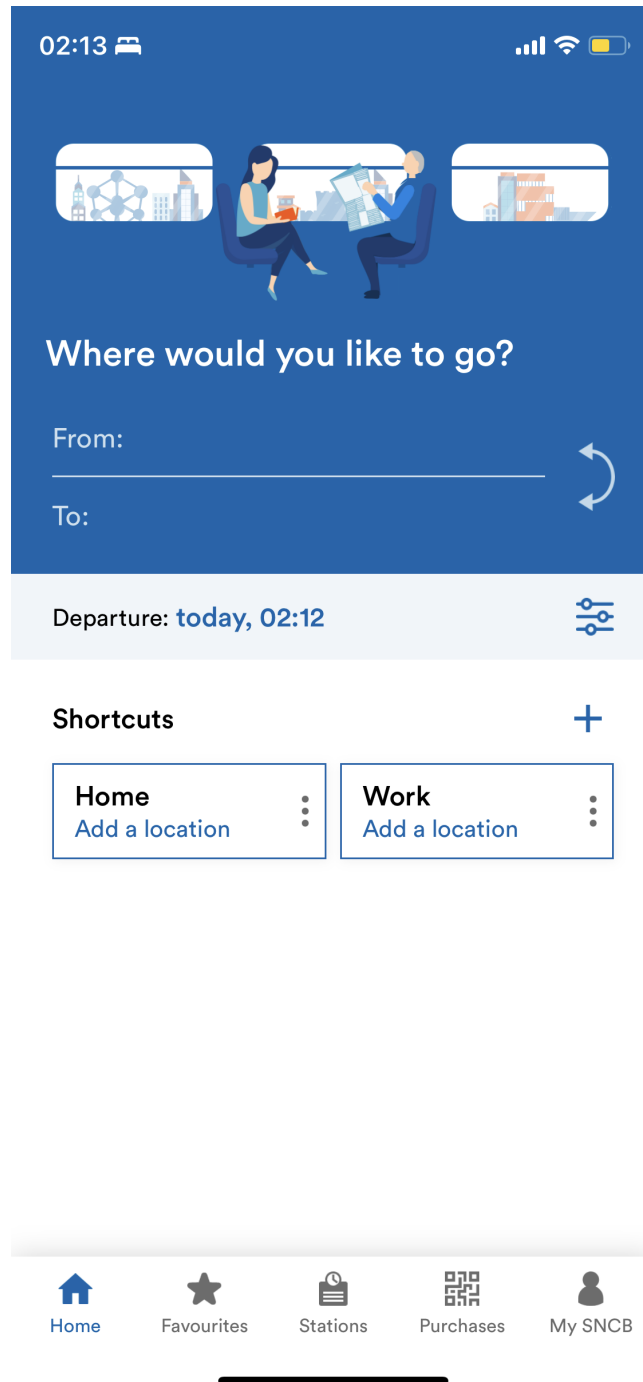


Figure 4.7: SNCB home screen

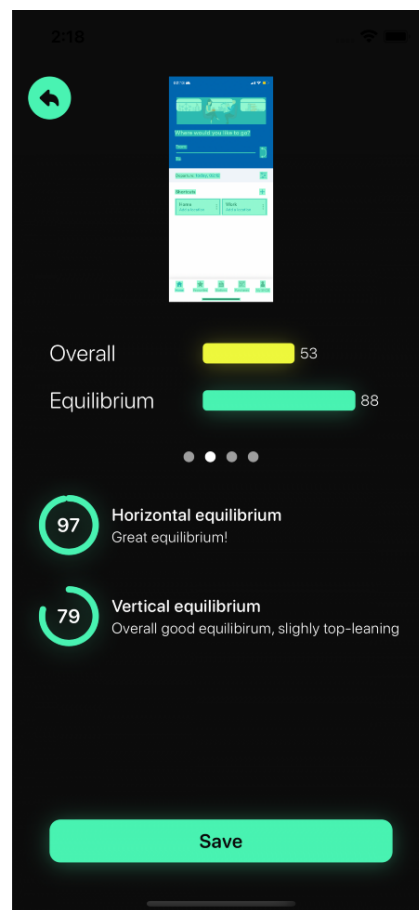
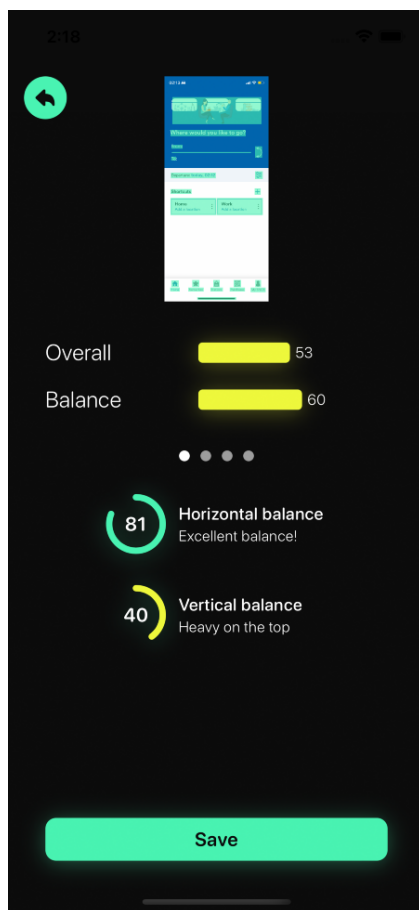


Figure 4.8: SNCB home balance and equilibrium ratings

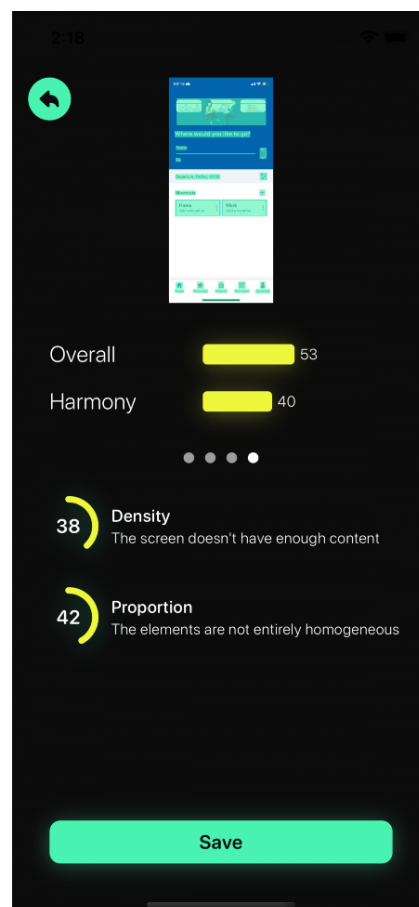
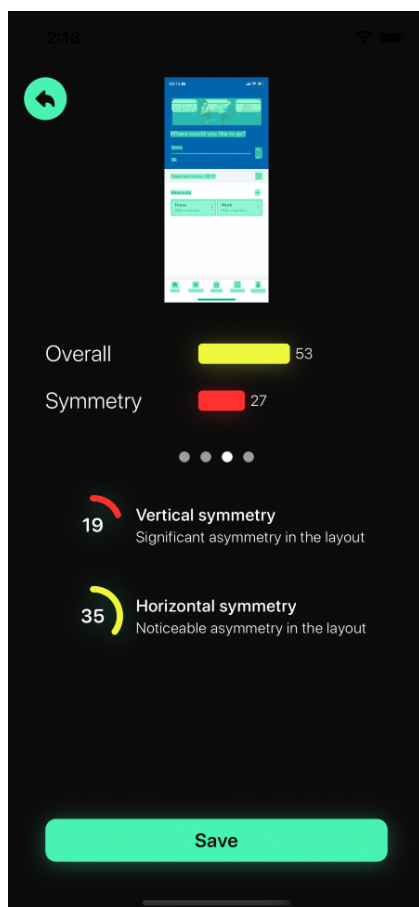


Figure 4.9: SNCB home symmetry and harmony ratings

4.2.2 Purchases

- **Horizontal balance:** the medium to high score can be explained by most items being fully centered and spreading equally across both parts of the screen. The few components that aren't though, like the "Active" and "Expired" navigation bar and the "Purchases" label, bring the overall balance score down. (*score: 62*)
- **Vertical balance:** the medium score is due to most elements belonging to the lower portion of the screen, with the upper one being relegated exclusively to a few small labels. (*score: 44*)
- **Horizontal equilibrium:** the very high score is justified, similarly to the horizontal balance, by most elements being perfectly centered and equally spread across the screen. The few that aren't, appear to have a very high degree of symmetry instead. The layout's horizontal center is therefore very close the screen's origin. (*score: 96*)
- **Vertical equilibrium:** because of the bigger elements being mainly around the vertical center of the screen, with its top and bottom edges being occupied by very small and minor components, the high score is to be expected. (*score: 86*)
- **Horizontal symmetry:** the motives for medium score are the same as the ones explained when analyzing the horizontal balance metric. (*score: 42*)
- **Vertical symmetry:** by careful inspection, we can notice that the screen's elements are somewhat symmetrical, therefore justifying the medium score. The "Active" and "Expired" labels are balanced by the "New purchase" button, the big central placeholder image is somewhat mirrored by the label right above it, but the tab bar doesn't have any specular component on the top part of the layout, apart from the small "Purchases" label, causing the overall rating to be lowered. (*score: 50*)
- **Density:** the low score can be explained by the screen being almost entirely empty, with only a handful of components present. (*score: 38*)
- **Proportion:** despite not many components being used in the UI, the few ones that are present don't appear to be entirely homogeneous, made exception for the tab bar at the bottom and the "Active" and "Expired" labels, which also have a somewhat similar aspect ratio. The medium score is then justified. (*score: 49*)

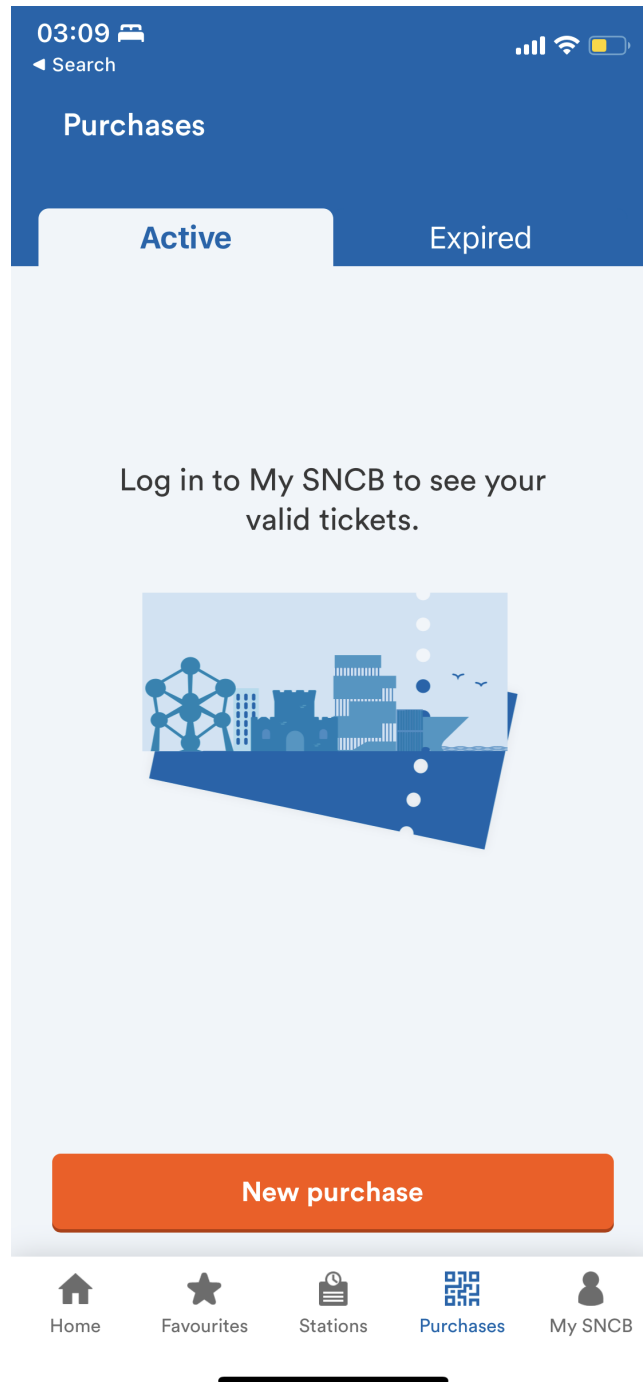


Figure 4.10: SNCB purchases screen

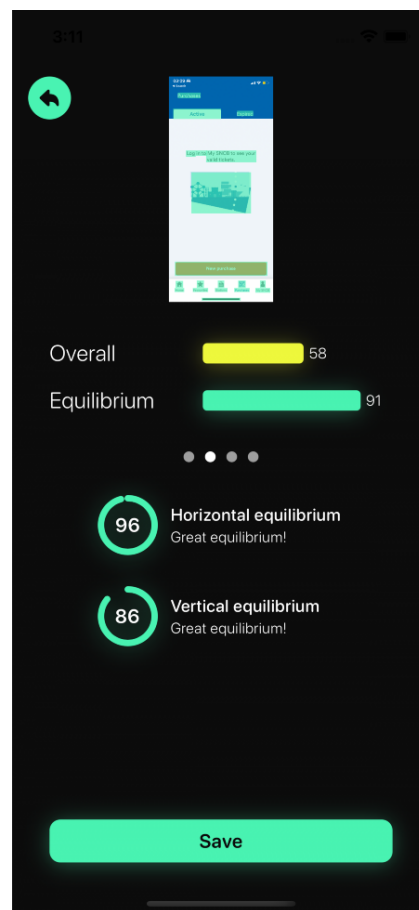
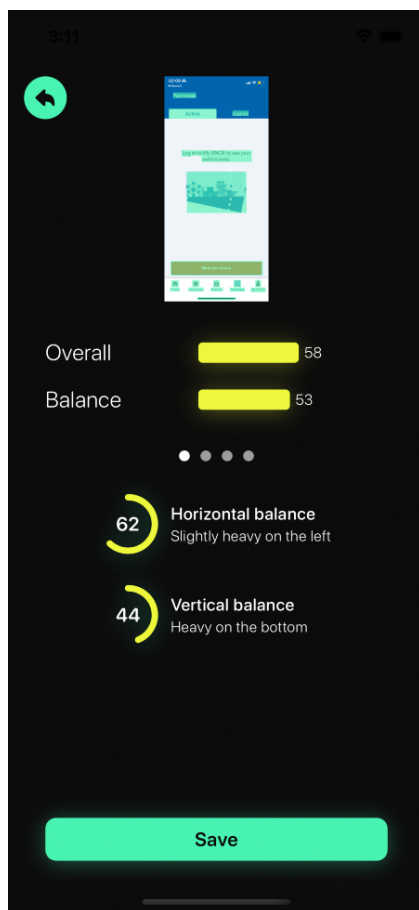


Figure 4.11: SNCB purchases balance and equilibrium ratings

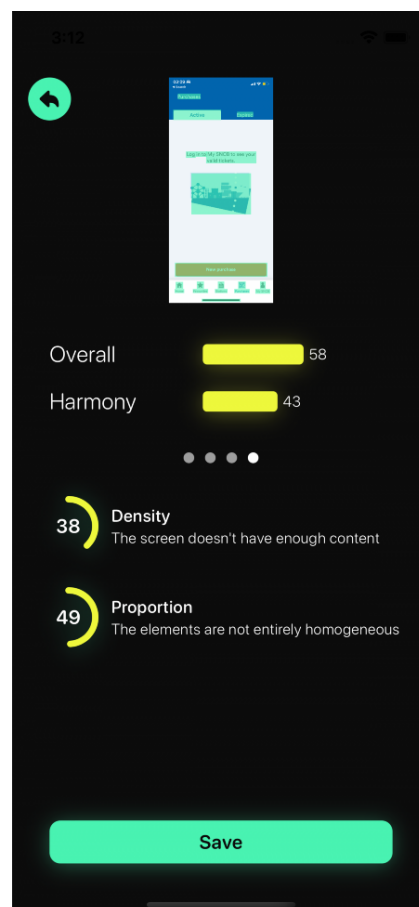
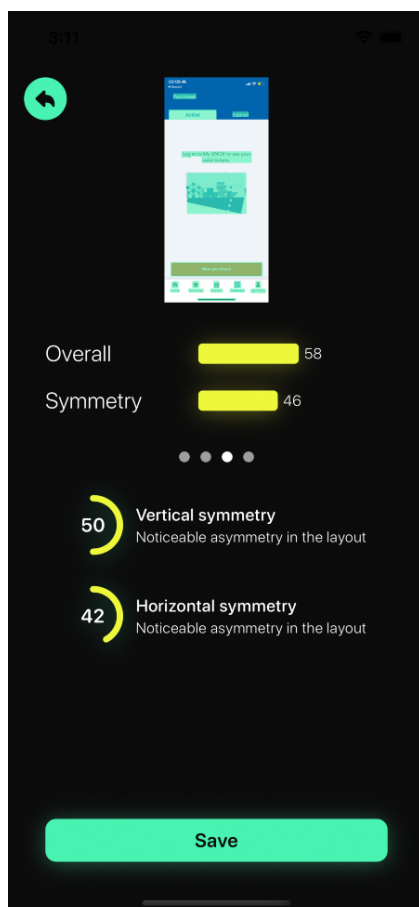


Figure 4.12: SNCB purchases symmetry and harmony ratings

4.2.3 Consistency

The home and purchases screen achieve a consistency rating of 88. This very high score could be explained by the fact that, although the two screens don't share many common elements, they still have common objects such as the tab bar, which is entirely identical, and are both rather top-leaning, with heavier elements concentrated around the center of the screen. Another contributing factor could be found in the scarce number of components in both screens.

4.3 Comparing results

While it was initially thought to be useful comparing the results obtained through Octodollop with similar UI rating service providers, such as Questim and AIM, both mentioned earlier on in this thesis, they turned out to be not fully functional when it came to the processing of mobile screens, with both of the platform failing to elaborate the given input, either fully (Questim), or partially (AIM).

It has been speculated that the difficulties that both providers encountered when attempting to grade mobile user interface images was due to the aspect ratio of the screenshots themselves being significantly different from those of a website, which was the main medium that both tools were built for.

The idea of embedding the mobile screenshots in wider, desktop-like pictures, with side margins filled by blank space, has been considered. This approach would have however introduced too many possible inaccuracies and inconsistencies in the final rating results and, for this reason, the idea of comparing Octodollop's metrics with other popular evaluation engines has been abandoned.

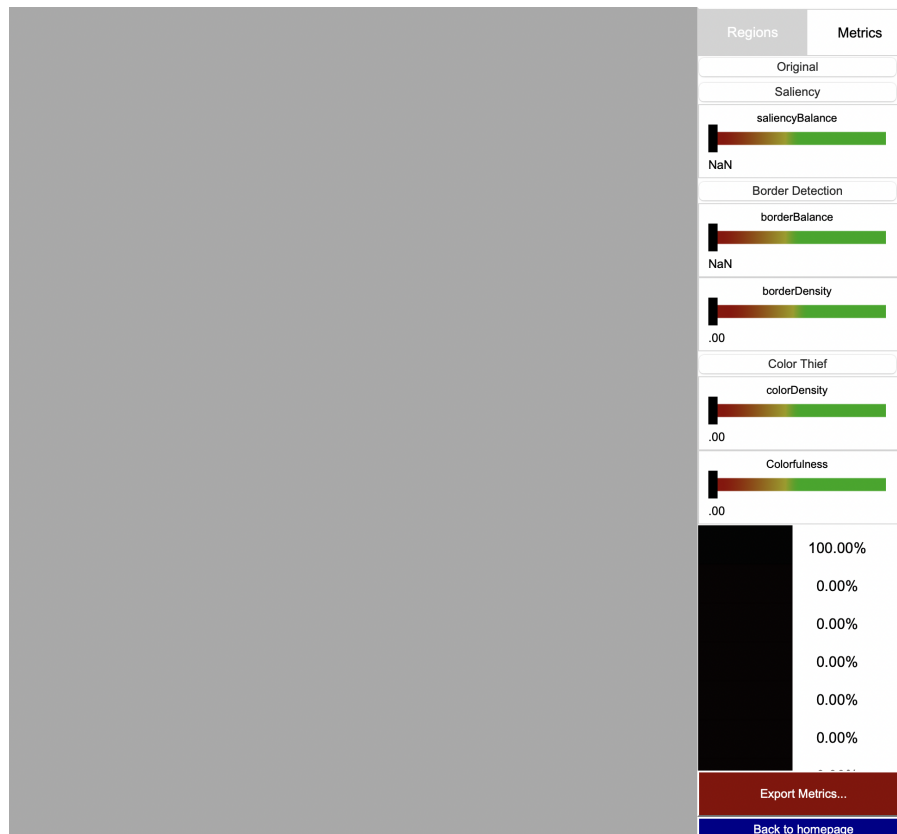


Figure 4.13: Questim, failing to elaborate the home screen of the Amazon iOS app

4.4 Conclusion

By analyzing and comparing the ratings of the two apps, we can evince some general rules about user interface design.

Screens that are largely comprised out of text, like the Amazon cart page, tend to have lower aesthetic scores. To improve this situation, one could try to enclose each element in its own box or card, like it has been done in the Amazon home page, where each of the "Keep shopping for", "Pick up where you left off" and "Buy it again" items are not simply placed on top of the screen's background, but surrounded by their own view component instead.

Interesting remarks can also be made about the use of screen's real estate. In the amazon cart app, for example, while an empty cart could have simply been shown with a single image and a text label, much like it has been done in the SNCB purchases screen, the remainder of the unused space has been filled with other components that, although not strictly related to the purpose of a typical virtual shopping cart themselves, contribute in making the screen appear fuller and more harmonious.

Alternatively, in case no additional content can be placed, one could opt for using bigger, near-full-screen images with bolder text labels, which would ultimately result in an increase of the density rating of the screen without fundamentally changing the purpose, scope nor usage of the UI.

Another way of increasing a user interface overall rating is by trying to use similar components throughout the entire screen. From what we have seen from the past four screens, the symmetry ratings were always rather low as the layouts were essentially divided into two significantly different top and bottom parts. Reusing similar components for both vertical halves of the screen will have the joint effect of increasing the proportion rating and, if placed consciously, increasing the vertical symmetry score as well.

Chapter 5

Final Conclusion

5.1 Overview

In Chapter 1, after analyzing the context of the problem and determining the importance of systematically assessing and measuring the beauty of user interfaces, we went over possible solutions to achieve that objective, coming to the conclusion that a mobile application was the best tool for our scopes and subsequently defining working hypothesis regarding the software, environment and target audience.

In Chapter 2, we went over the core concepts of our work: beauty and user interfaces. Through the studies taken into account and researched, the fundamental and undeniable impact of beauty in everyone's day-to-day life has been put under the spotlight and, after briefly going through the meaning of "User Interface" and its evolution throughout the decades, we tried to establish how the two concepts may interact with each other and overlap.

Chapter 3 was dedicated to the development of the entire infrastructure of the application. After a brief high-level overview of both the software and system architecture, the iOS app, the server (contextualised in the various aesthetic metrics computations) and the artificial intelligence border recognition model have been discussed in detail, illustrating their inner workings, formulas and algorithms employed.

In Chapter 4, the application was used to compute the ratings of two screens for two mobile apps, for a total of four screens and thirty-four aesthetic metrics. For each metric, its meaning, interpretation and possible root causes behind its score were gone through. Finally, the results were used to extrapolate some general conclusions about user interfaces design and practical advice that one could take into account in order to develop more pleasing, attractive UIs.

5.2 Future Work

There are multiple ways by which this work could be expanded onto broader horizons, some options and their possible ramifications will now be explored.

5.2.1 Adding more metrics

In order to offer a better experience and more point of views from which a user interface could be seen and rated, more aesthetic metrics could be computed. These would not necessarily need to be manually implemented, as already described in 3.4.7, leveraging instead existing services such as the "Aalto Interface Metrics service", whose publicly available APIs could be integrated into Octodollop's infrastructure.

5.2.2 Experimenting with different formulas

The formulas used to compute the various aesthetic metrics could be further on revised and adapted. The legacy implementations could then either be replaced entirely or be kept to offer a comparison on how different algorithms (which could be seen as different viewpoints) rate various aspects of a user interface.

As an example for experimentation, all the formulas detailed in 3.4 consider an element belonging to a plane's quadrant depending on whether its center falls within the quadrant bounds or not. The problem of elements segmentation could be researched and studied, testing different formulas that handle objects spreading across distinct quadrants of the canvas in different ways (e.g. splitting the "parent" element in as many elements as quadrants occupied).

5.2.3 Octodollop as integrated feedback gathering tool

The application could be expanded to, on top of simply computing the aesthetic score of a user interface, gather feedback from the user identifying the UI elements. While the basic infrastructure has already been put in place in the developed application to allow this kind of functionality, allowing the user to pick from a list of pre-selected comments after long-pressing on an identified element and communicating this information to the server, this option hasn't been further on explored as deemed too out of scope compared to the original purpose of this thesis.

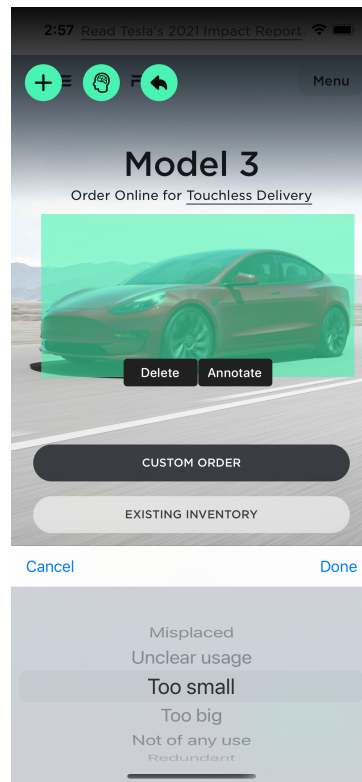


Figure 5.1: Octodollop in its current state, allowing the user to give feedback on the identified UI elements

List of Figures

1.1	Average monthly hours per mobile-app-user by age	1
1.2	Mobile app usage	2
1.3	The logo of the mobile application, Octodollop	4
1.4	Sequential phases of the validation & verification model	5
2.1	Perceived trustworthiness of people based on their beauty (adapted from [8]). . .	8
2.2	<i>Left:</i> the graphical user interface of the iOS operating system <i>Right:</i> a gesture-based interface, in which the user gestures are recognized and captured through the aid of a camera attached to a computer	9
2.3	<i>Top:</i> ED, an instruction-based text editor released in the 1970s <i>Bottom:</i> Microsoft Word, a widely used, GUI-based, modern text editor	10
2.4	The results of the aesthetic metrics computed by Questim on the Airbnb website	11
2.5	An example of the results obtained through the AIM platform	12
3.1	The interaction graph of the MVVM architectural pattern	17
3.2	The overall system's architecture	17
3.3	The Moore Finite State Machine representation of the iOS application	19
3.4	The class diagram of the core application's functionalities	21
3.5	The class diagram of the application's helpers	22
3.6	The server's class diagram (view layer not shown)	23
3.7	The home screen of Octodollop. <i>Left:</i> <code>ScreenInputViewController</code> <i>Middle:</i> <code>HistoryViewController</code> , saved UI ratings present <i>Right:</i> <code>HistoryViewController</code> , no saved UI ratings	24
3.8	The input preview screen of Octodollop. <i>Left:</i> <code>InputPreviewViewController</code> , when initialised with a URL <i>Right:</i> <code>InputPreviewViewController</code> , when initialised with an image	25
3.9	<code>InputPreviewViewController</code> displaying identified UI elements	26
3.10	<code>InputGatheringViewController</code> , Octodollop's input gathering screen, while an element is being identified	27
3.11	<code>RatingViewController</code> displaying a UI beauty score (only first three sections shown)	28
3.12	Popup in <code>RatingViewController</code> prompting the user to enter a name for the UI, shown once the save button is tapped	29
3.13	<i>Left:</i> <code>HistoryViewController</code> , selecting UIs to be fed into the consistency algorithm <i>Right:</i> <code>ConsistencyViewController</code> , displaying the consistency score	30
3.14	The coordinate system of a phone's screen	31
3.15	The vertical and horizontal balance equations, adapted from Ngo et al. [12] . . .	32
3.16	Indicative balance scores quality meter	32
3.17	The density equation, adapted from Zen [17]	33
3.18	Indicative density scores quality meter	33

3.19	The vertical and horizontal equilibrium equations, adapted from Ngo et al. [12]	34
3.20	Indicative equilibrium scores quality meter	34
3.21	The proportion equations	35
3.22	Indicative proportion scores quality meter	35
3.23	The vertical and horizontal symmetry equations, adapted from Ngo et al. [12]	36
3.24	Indicative symmetry scores quality meter	37
3.25	The results obtained with the final version of the iOS Vision-framework-driven border recognition model	38
3.26	The final algorithm used to obtain the results in fig. 3.25	38
3.27	The results obtained with the first version of the OpenCV model	39
3.28	The various steps of the utilised algorithm	40
3.29	The results obtained with the second version of the OpenCV model	41
3.30	The algorithm of the second version of the OpenCV model	42
3.31	Comparison of results before grayscale threshold adjustments (left) and after (right) on a user interface in "dark mode"	43
4.1	Amazon home screen	46
4.2	Amazon home balance and equilibrium ratings	47
4.3	Amazon home symmetry and harmony ratings	47
4.4	Amazon cart screen	49
4.5	Amazon cart balance and equilibrium ratings	50
4.6	Amazon cart symmetry and harmony ratings	50
4.7	SNCB home screen	52
4.8	SNCB home balance and equilibrium ratings	53
4.9	SNCB home symmetry and harmony ratings	53
4.10	SNCB purchases screen	55
4.11	SNCB purchases balance and equilibrium ratings	56
4.12	SNCB purchases symmetry and harmony ratings	56
4.13	Questim, failing to elaborate the home screen of the Amazon iOS app	57
5.1	Octodollop in its current state, allowing the user to give feedback on the identified UI elements	60

Bibliography

- [1] Ahamed Altaboli and Yingzi Lin. Investigating Effects of Screen Layout Elements on Interface and Screen Design Aesthetics. *Adv. Human-Computer Interaction*, 2011, 2011.
- [2] Apple. Foundations - Icons. *Human Interface Guidelines*, 2020.
- [3] Nicolas Burny and Jean Vanderdonckt. (Semi-)Automatic Computation of User Interface Consistency . 2022.
- [4] Michael D. Byrne, Scott D. Wood, James D. Foley, David E. Kieras, and Piyawadee Noi Sukaviriya. Automating Interface Evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 232–237, New York, NY, USA, 1994. Association for Computing Machinery.
- [5] Tim Comber and John Maltby. Evaluating usability of screen designs with layout complexity. *School of Commerce and Management Papers*, 1995.
- [6] Karen K. Dion, Ellen Berscheid, and Elaine C. Walster. What is beautiful is good. *Journal of personality and social psychology*, 24 3:285–290, 1972.
- [7] Camille Gobert, Kashyap Todi, Gilles Bailly, and Antti Oulasvirta. SAM: A Modular Framework for Self-Adapting Web Menus. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, page 481–484, New York, NY, USA, 2019. Association for Computing Machinery.
- [8] Yuanyan Hu, Najam ul Hasan Abbasi, Yang Zhang, and Hong Chen. The Effect of Target Sex, Sexual Dimorphism, and Facial Attractiveness on Perceptions of Target Attractiveness and Trustworthiness. *Frontiers in Psychology*, 9, 2018.
- [9] R. Mahajan and B. Shneiderman. Visual and textual consistency checking tools for graphical user interfaces. *IEEE Transactions on Software Engineering*, 23(11):722–735, 1997.
- [10] Aliaksei Miniukovich and Antonella De Angeli. Computation of Interface Aesthetics. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, page 1163–1172, New York, NY, USA, 2015. Association for Computing Machinery.
- [11] David C. Ngo and John Byrne. Another look at a model for evaluating interface aesthetics. *International Journal of Applied Mathematics and Computer Science*, 11(2):515–535, 2001.
- [12] David C. Ngo, Azman Samsudin, and Rosni Abdullah. Aesthetic Measures for Assessing Graphic Screens. *Journal of Information Science and Engineering*, pages 97–110, 2000.

- [13] Antti Oulasvirta, Samuli De Pascale, Janin Koch, Thomas Langerak, Jussi Jokinen, Kashyap Todi, Markku Laine, Manoj Krishnombuge, Yuxi Zhu, Aliaksei Miniukovich, Gregorio Palmas, and Tino Weinkauff. Aalto Interface Metrics (AIM): A Service and Codebase for Computational GUI Evaluation. In *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*, page 16–19, New York, NY, USA, 2018. Association for Computing Machinery.
- [14] Katharina Reinecke, Tom Yeh, Luke Miratrix, Rahmatri Mardiko, Yuechen Zhao, Jenny Liu, and Krzysztof Z. Gajos. Predicting Users’ First Impressions of Website Aesthetics with a Quantification of Perceived Visual Complexity and Colorfulness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, page 2049–2058. Association for Computing Machinery, 2013.
- [15] Sean Talamas, Ken Mavor, and David Perrett. Blinded by Beauty: Attractiveness Bias and Accurate Perceptions of Academic Performance. *PloS one*, 11:e0148284, 02 2016.
- [16] Enbar Toledano. May the Best (Looking) Man Win: The Unconscious Role of Attractiveness in Employment Decisions. 2013.
- [17] Mathieu Zen and Jean Vanderdonckt. Assessing User Interface Aesthetics based on the Inter-subjectivity of Judgment. page 7–10, 2016.
- [18] Ekin Çoraklı. On the beauty of nature, paintings and music. *International Journal of Human Sciences./ Uluslararası İnsan Bilimleri Dergisi*, 11:297–313, 2014.

Appendix A

Server

A.1 Sample requests and responses

Here follow a sample request and response for each of the server's endpoints.

A.1.1 `/rating` request

```
1 curl --location --request POST 'https://api.someurl.com/rating'
2 --header 'Content-Type: application/json'
3 --header 'Accept: application/json'
4 --data-raw '{
5     "items": [
6         {
7             "x": 0.567,
8             "y": 0.349,
9             "width": 0.5,
10            "height": 0.0249
11        },
12        {
13            "x": 0.198,
14            "y": 0.498,
15            "width": 0.789,
16            "height": 0.340
17        }
18    ],
19    "canvas": {
20        "width": 2350,
21        "height": 4509.32
22    }
23 }'
```

A.1.2 /rating response

```
1  "metrics" : [  
2    {  
3      "metrics" : [  
4        {  
5          "comment" : "Good balance, slightly left-heavy",  
6          "score" : 79,  
7          "type" : "balance_horizontal"  
8        },  
9        {  
10         "comment" : "Heavy on the bottom",  
11         "score" : 45,  
12         "type" : "balance_vertical"  
13       }  
14     ],  
15     "section" : "balance"  
16   },  
17   {  
18     "metrics" : [  
19       {  
20         "comment" : "Great equilibrium!",  
21         "score" : 97,  
22         "type" : "equilibrium_horizontal"  
23       },  
24       {  
25         "comment" : "Great equilibrium!",  
26         "score" : 86,  
27         "type" : "equilibrium_vertical"  
28       }  
29     ],  
30     "section" : "equilibrium"  
31   },  
32   {  
33     "metrics" : [  
34       {  
35         "comment" : "Significant asymmetry in the layout",  
36         "score" : 28,  
37         "type" : "symmetry_vertical"  
38       },  
39       {  
40         "comment" : "Significant asymmetry in the layout",  
41         "score" : 6,  
42         "type" : "symmetry_horizontal"  
43       }  
44     ],  
45     "section" : "symmetry"  
46   },  
47   {  
48     "metrics" : [  
49     {
```

```

50         "comment" : "The screen space is well used",
51         "score" : 90,
52         "type" : "harmony_density"
53     },
54     {
55         "comment" : "The elements are not entirely homogeneous",
56         "score" : 47,
57         "type" : "harmony_proportion"
58     }
59 ],
60 "section" : "harmony"
61 }
62 ],
63 "score" : 59
64 }

```

A.1.3 /ai/bounding_boxes response

```

1  [
2  {
3      "y" : 0.85837438423645318,
4      "height" : 0.14162561576354679,
5      "width" : 1,
6      "x" : 0
7  },
8  {
9      "y" : 0.78735632183908044,
10     "height" : 0.025451559934318555,
11     "width" : 0.15555555555555556,
12     "x" : 0.844444444444444444
13 },
14 {
15     "y" : 0.78735632183908044,
16     "height" : 0.025451559934318555,
17     "width" : 0.220444444444444444,
18     "x" : 0.071111111111111111
19 }
20 ]

```


UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl