Academy of Fine Arts and Design in Bratislava

RESEARCH AND ANALYTICAL TOOLS FOR TYPE DESIGN

ING. JAN CHARVÁT

fontanalyzer.app

Department of Visual Communication
Studio Typo

Assoc. Prof. Pavol Bálik ArtD.
Assoc. Prof. Zdenko Kolesár, PhD.

Bratislava 2025

CONTENTS

ACKNOWLEDGMENTS

This thesis would never see a daylight without many people that supported this project from the very beginning. My gratitude goes to:

Pavol Bálik for offering the very possibility of starting the project on AFAD, for all the support through the study and development process, for being the stable pillar the whole time.

Zdenko Kolesár for helping me during all 4 years to gain perspective and for all the reviews and comments on the project.

Filip Blažek for mediating first contact and for honest reviews that helped push this application further.

Domen Fras for changing my life path towards type design and education.

My fellow font engineers and type designers – Norbert Krausz, Bernd Volmer, Jiří Beran, Marek Čuban and Zuzana Konečná for testing, discussions and for keeping me on track.

My wife Zuzana Charvátová for everlasting support that is beyond imaginable.

Friends and family for their support and encouragement throughout the journey.

ABSTRACT

This thesis explores the work processes of Font Engineers (FEs)—the professionals responsible for the technical production of fonts. It aims to uncover the fundamental principles of their work, compile a clear and comprehensive checklist of common tasks performed by FEs, and develop an open-source tool offering essential tests that are freely accessible to everyone.

The first part focuses on research, drawing from existing literature and insights gathered from current FEs through an on-line questionnaire. The findings identify common production stages and the tests necessary to ensure technical quality of fonts.

Based on these results, the thesis proposes the design of an on-line application to deliver this functionality to a wide audience. The application will prioritize usability and adhere to established design principles, serving a diverse user base that includes students, designers, and, primarily, engineers.

The goal is not to create a complete replacement for FE work but rather to develop a tool that reduces the repetitive aspects of the job, making it less tedious for current professionals and more accessible to newcomers in the type design field.

KEYWORDS

fonts, font production, font engineer, quality assurance, web application, metadata, character set, language support, metrics, kerning, OpenType features, font family

1. INTRODUCTION

I have been working as a Font Engineer (FE) for approximately 10 years, and it's astonishing to see the sheer volume of repetitive tasks involved in our field, the inherent chaos in our workflow, and the formidable challenges faced by newcomers in the year 2023. Documentation is severely lacking, and many designers rely solely on the capabilities and standards defined by software developers. Up-to-date books addressing font engineering are nonexistent. For newcomers, the only viable approach is to learn from more experienced mentors within the field, with knowledge being passed down from person to person like ancient tales.

I am determined to address this situation through my PhD thesis, conducting research in the field of Font Engineering that will ultimately lead to the development of a new free automated engineering tool. This tool aims to assist Font Engineers, type designers, and beginners alike in producing higher-quality fonts. Furthermore, the creation of comprehensive documentation in this thesis and in the tool will make font engineering more accessible to a wider audience, ultimately benefiting everyone involved in the craft.

1.1. CURRENT STATE

The role of a Font Engineer is a highly specialized job title primarily focused on transforming already drawn typefaces into fully functional digital font files. "Already drawn" implies that the digitalization or creation of each glyph outline has been executed by someone else or, in some cases, by the same person responsible for producing the font. The Font Engineer typically lacks the authority to assess or modify the glyph outline unless explicitly requested or permitted to do so. Instead, the engineer's primary responsibility lies in configuring the font's metadata in a manner that ensures seamless compatibility with a wide range of applications and environments. This intricate process comprises numerous steps, each often following unique, individualized procedures. Unfortunately, these steps are neither systematically organized, adequately explained, nor publicly documented. As I will elaborate on later, every individual in this field tends to have their own distinct approach to the process.

The good news is that anyone with a computer can now create a digital font without any significant constraints. This accessibility has been the norm for nearly 40 years and has brought about significant changes in the industry. It's worth noting that this democratization of font creation means there are no restrictions related to licensing, hardware, or financial resources. Whether you prefer capable paid software or free programs with similar functionalities, you can use them on your preferred mainstream operating system.

This democratization has introduced a wealth of fresh ideas to the field, fostering amateurism in both the ideological and technical aspects of the process. Long-established procedures that have been known for centuries have been challenged since the public release of type design software, marking an inevitable part of the field's evolution. However, with this newfound freedom comes responsibility, and not everyone possesses the necessary education to handle it. This is one of the reasons why software like the one proposed in this PhD thesis is so necessary.

To put things into perspective, the largest font platform, MyFonts, currently boasts a library over 300,000 fonts (02/2025), and there could be an additional 100,000 available in other libraries, whether independent or open-source. This results in at least 400,000 fonts released into the wild. However, it's essential to ask how many of these fonts are technically up to date and error-free. Even within Google Fonts, despite relatively stringent entry requirements, many fonts pose usability challenges due to technical errors or oversights by their creators.

The early years of digital type design software were primarily focused on achieving the same level of quality output as their metal or photographic predecessors. The first digital fonts emerged in the 1970s and 1980s, and by the mid-1990s, the OpenType format became the industry standard. The most recent iteration, OpenType 1.9.1 from May 2024, defines the current state of technology in type design. This format is comprised of tables containing font data and metadata, with two flavors of font files, both adhering to the OpenType standard, but differing in outline construction and a few other tables. Modern software is capable of producing both formats seamlessly.

However, the real challenge lies in the human factor. While software can technically produce fonts without errors, it doesn't guarantee that these fonts will meet users' expectations. There are elements that software cannot judge, and only human intervention can address them. Like any production process, there should be quality control for font output. Some aspects are handled by font production software, but not comprehensively. Although there are many scripts, tools, and websites available to check fonts, there is no comprehensive list outlining what to check, what not to check, how to check, and what constitutes correct values or renderings. Often, these assessments come down to personal feelings, which may seem unusual in a fairly technical industry.

The best-known checking tool to date is Font Bakery, which was initiated in 2013 as an on-boarding process for Google Fonts and has been maintained by numerous contributors from foundries worldwide. Unfortunately, it is primarily available through the command line, limiting its accessibility to many type designers who may not be familiar with command-line usage or programming. Also, the input and output are only text based, thus not very visual. Every uneven line you must manually search for in the design software and decide if the issue is relevant or not. These limitations led to the concept of creating a new visual, highly user-friendly tool without such prerequisites.

I believe having tool that is available for everybody, free for use, with testing capabilities, easy to use, will help produce better fonts overall. Many of the designers would like to take more care but at the moment they just don't know how. I've discovered the workflows and tools only being employed at biggest font production company ever. Without that I would hardly know any, and I may even not know I need some.

1.2. GOALS

1.2.1. Primary objectives

Primary goal of this thesis is to gather elemental information about Font Engineering and together with my experience create free, easy to use testing environment.

In First phase I will try to accommodate as much information about the process as possible. There are few sources that I want to try dig into:

- OpenType specification has on-line documentation, which defines every field used in font file, gives it meaning and intended usage.
- Other open-source testing tools can lead me to new tests or processes.
- Questionnaire, where I ask other font producers (engineers, designers, quality assurance) about their processes.
- There are two books Karow, P. (reprint 2011, original 1994), Font Technology and Haralambous, Y., & Horne, S. P. (2007) – Fonts & Encodings: From Advanced Typography to Unicode and Everything in Between – both can give mi some insight to font background and testing procedures used in their times.
- Own experience is my last source. I will try to avoid this option as much as possible, because the result should be based on some exact scientific principles and not personal opinions and favorites.

Next, I aim to compile a comprehensive list of essential phases that must be undertaken for each font under examination. Each phase will necessitate a specific collection of tests. During this phase, my objective is to maximize the depth of research by cataloging every conceivable test, even those for which I may not possess the immediate capacity to develop code, whether due to time constraints or knowledge limitations. It is important to emphasize that this encompassing approach extends not only to individual fonts but also to font families, typically comprising two or more fonts. Consequently, I must document tests tailored to both individual fonts and font families.

From my Font Engineering practice I got to know there are few different personas in the type industry with different knowledge and approach and there is not a single language you can talk to them. Therefore this application must be able to differentiate the to different types of users. You simply can not show students all the tables in the font, that would distract them if not discourage them from using the app ever again. For more advanced users the app should not oversimplify the results for the same reasons.

As this research progresses, critical decisions must be made regarding the framework's deployment strategy and the selection of a programming language for code development. While I appreciate the reasoning behind the command-line design of Font Bakery, my goal is to bring the framework closer to end-users. Font Bakery primarily serves as an integral component of the font on-boarding process, rather than a tool intended for broad public utilization. Therefore, my aspiration is to ensure universal accessibility, accommodating users with diverse operating systems, by providing an on-line tool accessible through standard computers equipped with web browsers and Internet connectivity. Moreover, the framework will adopt an open-source ethos, facilitating ongoing collaboration, code improvement, and functionality expansion. The core features of the tool will remain freely accessible, with a strong emphasis on user experience (UX) and adherence to principles of usability. Rigorous testing will be conducted using actual fonts and real users to validate its practicality and effectiveness.

1.2.2. Methods

To gather pertinent information regarding font testing, I employed several methods. As previously discussed, the challenge lies in the scarcity of up-to-date sources that offer a comprehensive view of font testing. However, this scarcity of resources underscores the significance of the work being undertaken.

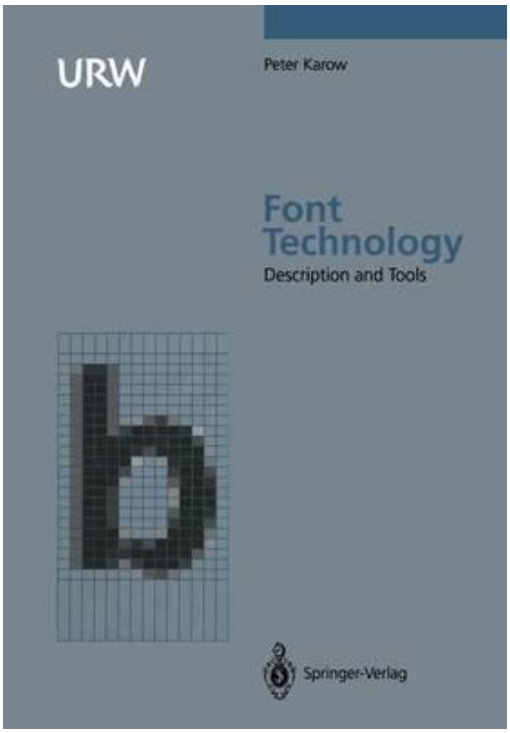
2. PRELIMINARIES

2.1. RESEARCH

2.1.1. Books

Initially, my efforts were directed towards gathering books on the subject matter. Regrettably, I discovered the absence of dedicated books specifically focused on Font Engineering or Font Testing Quality Assurance (QA). Nevertheless, there are two books that delve into the technical aspects of fonts.

The older of the two, "Font Technology: Methods and Tools" by Karow, P. (reprint 2012, original 1994), has proven to be an invaluable resource. This book aids in the elucidation of historical processes within type design, particularly during the era of hot metal type, industrialization, and photo typesetting. In those bygone days, type design involved the physical creation of numerous metal objects to facilitate the setting of lines of text or the production of books. Remarkably, some of the principles established during that era continue to influence contemporary font technology. The book also offers insights into the early stages of digitalization and the development of tools that served as precursors to the digital technology we employ today.



Book cover of Karow, P. (2011). Font Technology: Methods and Tools

Within the pages of this book, one can find a definition of the current topology of font curves and the underlying principles used to describe such shapes. In retrospect, the decision to adopt Bézier curves as a resolution for font curves appears to have been a prudent choice, as this technology remains relevant to this day. While it's acknowledged that some designers now find Bézier curves limiting, these individuals are often pushing the boundaries of technology. It's conceivable that in the coming years, a new method for describing curves may emerge to cater to their evolving needs.

As I delve deeper into the book's content, I am struck by the fact that pioneers of that era grappled with many of the same challenges we continue to face today. While some of these challenges have been surmounted through the establishment of higher standards, others persist as ongoing hurdles in our field, prompting us to explore innovative solutions and advancements in font technology. Kerning is one example of them. Kerning means that you have a pair of letter-forms, let's say A and V that to be set properly next to each other, their

squares need to overlap each other. That bothers people from early type design days. In this book there is a description of an algorithm that could solve the problem, but it is never a perfect solution. Till today there are like dozen of algorithms trying to achieve the perfect balance of blacks and whites in the process of font development, but none of them achieved superiority. Today technology of big data could come handy using thousands of fonts having "correct" or "human proved" kerning pairs and distill the values for new font from few settings put in that algorithm. Of course that does not solve the problem with new shapes, or new scripts or alphabets, it only solves the problem with current settings we use and have enough examples.

example of kerning

One of the most interesting parts of this book at least for my future PhD. work seems to be the 11th chapter called Type Quality. It discusses quality from many perspectives. Design or form is the first and obvious one. There are many type design principles proven by time, by the way human vision works or by general design principles. Are straight lines really straight? Is on-curve points distribution well? Are they in extremes? Is the balance of off-curve points well made? Are Metrics consistent? Are kerning classes senseful? Are there basic kerning pairs introduced? Are there OpenType features? Are all glyphs accessible by feature or by Unicode? Does the baseline position the text in the middle of a text line? "Our discussion will reveal that the craftsmanship of a typeface can be defined and specified, and thus also measured."[1] But none of them assure you as a designer, that the font you design is good quality font and it will be well sold. In the end of the chapter this publication goes even to evaluating these qualities and give each criterion its weight adding to 100.

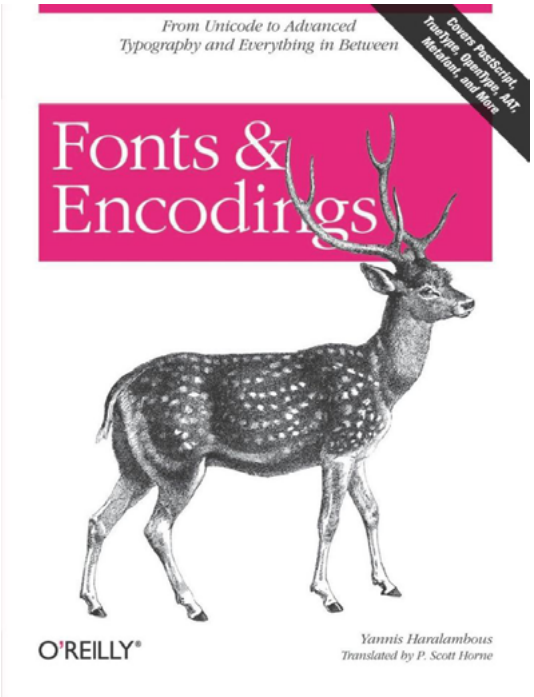
Another compelling subject covered in this book is legibility. The author draws upon a wealth of older and more recent studies, providing valuable insights that address key questions for designers. Some of these findings include:

- Research by Paterson and Tinker in 1929 indicated that a 10pt font size is optimal for readability.
- According to Tinker and Paterson's study in 1949, a minimum of 2pts leading is required for legibility, and an additional 2pts can boost readability by 3.24 times.
- A study by Paterson and Tinker in 1932 found that sans-serif typefaces are equally readable compared to typefaces with serifs.
- Their research from 1928 revealed that uppercase letters are 11% slower to read than lowercase letters.

1 Karow, P. (2012). Font Technology: Methods and Tools (Softcover reprint of the original 1st ed. 1994 ed.). Springer.

- Regarding italic, slanted, or oblique typefaces, they found that these typefaces are read at the same speed when used for short text portions. However, for longer reading, the results deteriorate, with a potential decrease in reading speed of up to 6.3% over a 30-minute reading period.

The second book, authored by Haralambous and Horne in 2007, titled "Fonts & Encodings: From Advanced Typography to Unicode and Everything in Between," offers a comprehensive overview of how text is stored in computers. This exploration spans the evolution of text encoding standards from as early as 1931 up to the contemporary standards of 2007. The book primarily centers on the intricate topic of Unicode and the storage and display of text from diverse regions of the world.



Book cover of Haralambous and Horne, 2007. Fonts & Encodings: From Advanced Typography to Unicode and Everything in Between

The simplicity of encoding and displaying our Latin alphabet on screens is highlighted. In an era marked by constraints on the number of available glyphs for font designers, engineers devised code-pages, providing a manual for coding and decoding files. This may resonate with experiences like watching a movie with Eastern European subtitles but encountering encoding issues that result in the display of incorrect characters. With advancements in storage capacity and memory, Unicode emerged, offering distinct ranges of glyphs for each script, such as Latin, Greek, Cyrillic, Georgian, Korean, Chinese, Thai, Vietnamese, and even emojis familiar from various messaging apps. It's worth noting that as of that time, nearly 150 scripts were not yet encoded into Unicode, indicating ongoing work in this area.

While this aspect of text encoding touches my work only peripherally, the book's extensive content, spanning over 1000 pages, offers valuable insights. Of particular interest to me are the appendix sections, which delve into the technology behind curves, hinting, and OpenType features. These resources serve as valuable references as I refine my understanding of what QA entails in the context of font engineering and testing.

2.2. QUESTIONNAIRE

Another method I've employed is to gain insights into how fonts are tested by other font engineers and type designers. To facilitate this, I've developed a concise Font Quality Assurance questionnaire. This approach allows me to collect contemporary and pertinent information about the current state of testing methods, tools, and resources in the field.

2.2.1. Contents

Overall information

Your field / Job Title / What do you do?
How you consider yourself? (I know independent designers do all of that, but let me understand what is your expertise)

- Type Designer (TD)
- Font Engineer (FE)
- Quality Assurance (QA)
- TD/FE/QA (unspecified font related work, ie. touching fonts)
- Management/Sales/Marketing (related to fonts, but not actually touching fonts)
- Other...

Independent/Foundry / Employee

- Major Foundry (20+ people)
- Independent Solo / Independent Foundry
- Employee of a Foundry=

How many fonts you do produce per year (not only for yourself, overall)

- 1–5
- 6–10
- 11–20
- 20+

How do you consider your skills? (points like stars 1-least true, 5-most true)

- I do type design with ease
- I understand concept of metrics / kerning
- I understand anchors and building diacritics
- I know what features do
- I can setup a font before export
- I know some font standards
- I know how to test fonts in apps (InDesign, Word, etc...)
- I know how to test fonts in command-line
- I know how to test fonts in browsers

What Font software you use

- Glyphs
- Fontlab
- Robofont
- FontForge
- Other

QA specification (second part)

Do you use any kind of QA to test your fonts?

By ANY I mean literally anything, even trying to install the font is kind of a QA, or opening InDesign and seeing the styles is some sort of QA.

- Yes
- No

Do you test your fonts before you export them from Font Software or after?

- Only in Font Software (Before)
- After Export
- Before and After

Do you use internal / external QA tools?

- Internal (tools developed by you or company you work for, not considering Font SW)
- External (tools developed by other people, including Font SW)

What do you test?

HINTS: Design QA (diacritics, outline errors, alignment zones, mixed glyphs, ...), Charset, (glyph order, ligatures, duplicates, spaces, hyphens, unicodes...) Font Info (masters, stems, interpolations, style linking, instance naming, italic angle, vertical metrics, ...), Metrics, Kerning, Features. After export: InDesign, Word, style naming, style linking, features working, kerning working, hinting, ...

- Text Answer only

What would you like to test, and you don't have the tool or the knowledge or the time to do so?

- Text Answer only

What QA tools do you use?

name, authors, web address pretty please

- Text Answer only

Do you know any nice tool, that is not necessarily QA, but it is useful or pleasant work with in regards of fonts?

- Text Answer only

In a broader view Is there a functionality that you miss in current font files?

- Text Answer only

And related to last question is there a functionality / table / process which you find obsolete / redundant / not useful today?

- Text Answer only

QA Tool (third part)

Would you be interested on a new website giving you QA testing in one place or you feel comfortable with your tools?

- Yes, I would at least try it
- No, I'm fine with what I have at the moment

Would you be interested in a publication/wiki explaining font standards and QA testing?

- Yes
- No

Would you be interested in Foundry Specific testing of fonts (metadata, charsets, etc...)

- Yes
- No

There could be several levels of user interaction. What level you think would be appropriate for you?

If you choose other option, please describe your expectations

- Easy / Beginner (only essential must have checks, so the font is not broken, easy explanations, not using much of a slang)
- Intermediate / Advanced (more checks, deeper explanations, use of some slang)
- Expert / Professional (all checks, full explanation, specification citations, use of slang, on server checks, downloadable results of external tests)

Where would you prefer to use the tool?

Fonts on the server will be deleted after testing anyway.

- Upload font to server / server/side = more checks
- Browser side check, no upload / local = less checks
- I would install it on my/our server and test it locally / self-server

Do you miss some kind of test or functionality related to QA? Anything that would make QA easier?

- Text Answer

2.2.2. Results

I received over 60 responses from font foundries and individuals worldwide, which provided invaluable insights into current font testing practices. The respondents predominantly consisted of type designers, with 85% working independently, and 90% producing between 1 and 5 fonts annually. A significant majority, at 65%, exhibited a high level of skill in type design. Moreover, a substantial proportion of respondents possessed proficiencies in areas such as metrics and kerning (83%), working with anchors (81%), and utilizing font features (76%). Approximately 61% could prepare fonts for export, while more than half (52%) have knowledge of some industry standards. When it came to font testing environments, slightly over half (61%) conducted tests in applications like InDesign, but only a minority (23%) were acquainted with command line testing, and fewer still (40%) were familiar with browser-based testing. Notably, 82% of respondents indicated using Glyphsapp as their preferred tool.

The next segment of the survey delved into testing practices. A staggering 93% of participants confirmed that they engage in font testing in some form. A significant majority (73%) conducted tests both before and after exporting fonts, demonstrating a comprehensive approach to quality assurance. Furthermore, 55% of respondents reported having established internal testing procedures, highlighting their commitment to quality control. Nearly all respondents (90%) made use of external testing tools to augment their testing efforts. The specific aspects of font testing and the corresponding outcomes are detailed in the following table:

STAGE	TEST
META	does the font match the OpenType spec
META	style linking
META	stems
META	italic angle
META	underline thickness and position
META	strikeout position and size
META	alignment zones
META	weight class
META	match the names the proper weight classes
META	width class
META	use Typo Metrics Custom Parameter
CHARSET	charset
CHARSET	stylistic sets have friendly names
CHARSET	glyph order
CHARSET	ligatures

STAGE	TEST
CHARSET	duplicates
CHARSET	spaces
CHARSET	hyphens
CHARSET	unicodes
CHARSET	all Tabular glyphs have same width
CHARSET	glyphs that are neither accessibly via Unicode or feature
OUTLINE	outline errors
OUTLINE	outline direction
OUTLINE	kinks
OUTLINE	overshoot consistency
GLYPH	base glyph is first
GLYPH	composites with same base glyph have consistent width
GLYPH	diacritics
GLYPH	mixed glyphs
GLYPH	rotated and scaled components
METRICS	metrics
METRICS	vertical metrics
KERNING	kerning
KERNING	kerning: consistency among masters
KERNING	touching Glyphs (kerning?)

It covers quite big amount of I think can be tested. Next question covers testing tools which goes into this tab:

https://wakamaifondue.com/
https://www.setuptype.com/x/fontinspector/
https://fontdrop.info/
https://fontgauntlet.com/
https://underware.nl/latin_plus/character_set/
http://vmt.dizen.cz/
https://opentypecookbook.com/
http://www.cyreal.org/Font-Testing-Page/
https://www.alphabet-type.com/tools/charset-checker/
http://ultrasparky.org/archives/2014/03/spacing_tests.html
https://vertical-metrics.netlify.app/
https://lorp.github.io/samsa/src/samsa-gui.html
https://hyperglot.rosettatype.com/
https://www.motaitalic.com/tools/font-tester/latin/

RMX tools in Glyphsapp
Kern On in Glyphsapp
Speed Punk in Glyphsapp

Missing Functionality**:** Respondents expressed a need for certain missing functionalities, including security features, Higher Order Interpolation (HOI), automated kerning generation, and greater automation in general.

Obsolete Features: On the flip side, when asked about obsolete features, the most common responses were legacy marks, vertical metrics, and PANOSE.

Interest in New Tools: An overwhelming 97% of participants expressed a desire for a new font testing tool, while 98% indicated an interest in a publication that explains Quality Assurance (QA) and font testing. Additionally, 90% of respondents expressed interest in the ability to test custom metadata or charsets in foundry-specific tests, underscoring the importance of tailored QA solutions for specific font foundries.

Testing Levels: A majority (60%) of interviewees expressed a preference for using the new tool at the Expert level, with an additional 30% comfortable at the Intermediate level. Some respondents also emphasized the importance of explanations accompanying test results, suggesting a more sophisticated version of the tool where each test is elucidated and supported with external links for reference. This insight hints at the possibility of streamlining the tool's test versions, potentially offering only beginner and expert levels to cater to users' needs.

Testing Locally vs. Sending Fonts: A critical question revolved around whether users would prefer to send fonts to a server for testing or conduct tests locally. Roughly half of the respondents would be willing to send fonts to a server, while the other half displayed reservations, likely due to common practices of signing Non-Disclosure Agreements (NDAs) with clients to safeguard intellectual property (IP). Interestingly, only a small fraction (17%) indicated a preference for a browser-based version, leading to the decision not to invest further in this option. Furthermore, 33% of respondents expressed an interest in installing the app on their own servers, highlighting strong support for an open-source approach to the project.

Overall, the questionnaire has played a pivotal role in aligning the project's goals with the needs and preferences of users. It has informed critical decisions about the tool's functionalities and test procedures, ensuring that the final product meets the demands of font engineers and type designers effectively.

3. APPLICATION

3.1. REQUIREMENTS

To succeed with development of an application, I need to set realistic requirements, which meet with aim of this thesis and given time.

Here's a list of features that could be the base of the new application:

- Any computer architecture or operating system (OS) or font development app
- Based on existing frameworks
- Easily extensible
- Free to use (uncustomized version)
- Able to test multiple OpenType fonts
- Able to test OpenType font families
- Usable (UX)
- Tested on real exported fonts
- Tested by real FEs

The first point mentioning any OS or development app is very important. There are few of type design applications out there running on few different operating systems. That makes a mixture that would easily employ whole room of engineers. I'm not native programmer in any environment and it would be extremely hard for me to get to know all the different specifics of releasing an app in these systems or apps. Developing plugin would also limit me to current state of the type design application set and it would be hardly extensible later when updates or new apps will be coming to the market. Also, I want to mention there are already on-line apps for type design and there could be also difficult to integrate this app into. To summarize this, point this app should be something not connected to current type design applications and rather be independent of any specific environment. This will also make sure that it will be accessible to biggest part of the type industry.

For the sake of simplicity, all mobile devices will be ignored during development because there are no fonts accessible on them and all of the font design is happening on desktop computers. If this situation will change in the future, the logic can stay the same, only visual hierarchy of the presentation will need to change.

To define where it should be deployed, we need to find an environment which is architecture, OS and app independent – the only one available is World Wide Web. Web environment is independent, up to date and well maintained by many groups of developers from many companies. This ensures also stability and longevity of this app compared to other development environments. Web is accessible from any location (theoretically), any OS or app and all the developers try to stick with its standards. There is great support for current font formats and their features, thus no need to write extra code to make something work.

This app should be based on existing framework is a logical point, where I can only state that there are few frameworks that can handle fonts and most of them can extract all the needed data from the font files. So, there is no need to

write own framework or library, that would allow us to access properties or functions of the fonts. Also, that would not be the aim of this thesis.

Here are the options I've researched:

- Free Type (original authors: David Turner, Robert Wilhelm, Werner Lemberg) project written in C++ which could be used and deployed on the webpage. I personally have zero experience using FreeType, I only use TTF Autohint as a user of Glyphsapp (which is part of it) and my knowledge of C++ is fairly outdated. It is well maintained and well written library that focus on displaying font in C++ applications. This approach would lead to server-side application, where users would have to upload their fonts to the server.

- Opentype.js (author: Frederik De Bleser) is a library written in JavaScript. That was a serious option to consider, but at the end I did not choose to pursue it. It has a great advantage of inspecting fonts in a window of a browser; therefore, nothing is stored on the side of the server, which is important for many designers working under NDA for other companies. I will address this issue later on in this chapter.

- FontTools (author: Just van Rossum) is a python library for reading, manipulating and rendering fonts. It is well maintained open-source library used on many type and engineering projects. I'm quite familiar with this library and with Python as a language so this seems and an ideal choice for this application. Running python on a webpage is not native and you have to use framework as Django to make it work.

To get back to the server-side / client-side option to choose from. The state of the font industry is that there is an over production of fonts. There are many more fonts that market would ever need, many duplicates or near duplicates of famous fonts and so on. I don't want to discuss here why there should be more fonts, but I can safely say that there is always some new problem which could be solved by a new font and this solution is much more often a custom font. Custom fonts means that the designer creates font family unique to the client's needs and that usually involves NDA to be signed there. In NDA usually states that no other persons or system should be able to access the font files, and this would effectively prevent the designers from using server-side applications.

Client-side application has an advantage of being completely run in a client's window and not sending any data to the server. Regarding last paragraph that would be better option to pursue, but I decided not to go this way. Previously I've tried to build such an app with opentype.js as a test of feasibility. It was working nicely, but at some point, I was thinking that this could not take advantage of server capabilities. There is no way of incorporating this into larger projects, custom test would be much harder to implement, and I would not be able to run any python scripts or tests that already exists. Also, the computation time would be heavily dependent on performance of client's computer. The choice here was not easy, but the reasons behind server-side were stronger as a long term solution – user profiles, test history, bigger scalability, FontTools and Python libraries and possible integration / customization for clients. There may be legal possibility to securely rent users piece of shared storage space and avoid the problem with data privacy and NDAs.

Another feature listed is to be easily extensible. That means if anybody wants to add their own functionality, they can do it easily. Also, that means as developer I can easily incorporate new functionality if available for example in any library. To be able to run custom tests we need to define how this may work. Going to the core of the meaning, that would suggest that users could write their own

test function and add it to the testing procedure. This would be possible, but quite hard to implement. I believe most of the users do not wish to write their own functions, but they may want to specify the parametres of the already existing functions. There may be few that would like to have more insight in the process and maybe adjust some values or define own lists to test from their own test routines. This is the custom that is feasible and easy to incorporate. The user will only provide human readable data to the system, and it will be saved to database/filesystem and run in his own account. This is also good reason to choose server-side system.

It must be tested or exported fonts. Testing source files in design app is very important during design phase or when transitioning to the production of the fonts. The issue here is that the results of export of the same file can vary from different applications or even different versions of one application. So, testing the source file is not reliable way of testing the fonts or families. Clients receive exported font files, not source files (usually), so final testing must focus on these deliverables..

There are 2 points addressing multiple fonts and font families. The app should support batch testing, allowing multiple font files to be analyzed simultaneously. Testing only single file would be very time consuming and it would not allow to compare the styles between each other. Testing a font family gives the designer another level of overview on the font file set. There are certain parameters that has to be aligned though all the files in the family and this needs to be tested and presented to the user as well.

It has to be tested by real Font Engineers. That would be an ideal state of development, but with the list of e-mails from questionnaire I'm fairly confident that I will find at least few engineers to test it with. I'm also a font engineer, but my opinion on the solutions is biased and I may not see better solutions because I could be stuck in local minimum and not be able to see better solutions. I will regularly consult the process with fellow font engineers to have some sort of feedback. I do not assume they will spend a lot of time testing it, but even few remarks can drastically help me move forward.

This is connected to the UX part, where it states it must be usable. That is slightly vague definition, because there is no way we can say something is usable. We only can state that is not usable, when the expected action is not happening. It could be than evaluated as negation of the not usable state, but that is not correct it terms of usability. We could take a definition from usability book: "Highly usable web sites are intuitive. They are transparent. They support the users and allow users to accomplish their goals quickly, efficiently, and easily. For example, one aspect of usability is that users should know what to do next. They should either be given explicit instructions or the web site should follow some known interaction pattern."[2] We can also find a more recent definition: "Good website usability means that a website is intuitive, meaning that users can navigate it without having to consciously think about the navigation process. It also means that the information presented is clear, the website is reliable and performs well across various devices and browsers. In essence, web usability is about creating a seamless, straightforward, and engaging on-line experience for users."[3] For this feature there are good conditions. All the web-browsers are rendering content very much the same and developer can rely on standards be-

2 Brinck, T., Gergle, D., & Wood, S. D. (2002). Designing Web sites that work : usability for the Web. Morgan Kaufmann Publishers.

3 Website Usability: The Ultimate Guide for 2024. (n.d.). Survicate.com. https://survicate.com/blog/website-usability/

ing relatively well implemented. To ensure seamless user experience, designer should start with sitemap and wireframes. “Wireframes are basic visual representations of a user interface that outline the structure and layout of a webpage or app.”[4] Big role in designing good experience on the website is continuous testing. As mentioned before I will conduct user testing with real font engineers as often as possible.

The application must be free for users. Here it was never a decision to me, but let’s discuss both possibilities. Making this application as a paid service only is definitely one of the possible ways. Having money can bring better performance on the server side, allocating dedicated server with more computational power. It can pay developers constantly developing new test and improving the user experience of the service. It would be completely valid approach. But one must also see the context this is emerging in. There are competitors as discussed before like Font Bakery that can test many font properties for free, there are also many web-sites that do testing of one specific feature. Also many of the properties could be tested in the design software already during development. So why to pay for a new service, offering similar functionality as free competitors? Would it attract enough users to make itself sustainable? I don’t know. But making it fee based will discourage most of the uneducated, that could benefit the most from this app. And helping the broad public, enthusiastic hobby designers, lone wolfs was always the my aim on this project. Also I think there could be another way of making this application self sustainable, but still free for all the users. Let’s make the most advanced users pay for customization and advanced features.

I am optimistic that I can implement most of the envisioned features successfully.

3.2. INTERNAL STRUCTURE

The application will have to have 2 layers where different users with different roles will interact. Django offers effective way of connecting these two together.

Layers:

- Frontend
- Backend
  - Python scripts
  - Filesystem
  - Database

“In software development, frontend refers to the presentation layer that users interact with, while backend involves the data management and processing behind the scenes. In the client–server model, the client is usually considered the frontend, handling user-facing tasks, and the server is the backend, managing data and logic. Some presentation tasks may also be performed by the server.”[5]

4 Interaction Design Foundation. (2016, September 25). What is Wireframing? The Interaction Design Foundation; Interaction Design Foundation. https://www.interaction-design.org/literature/topics/wireframe

5 Frontend and backend. (2021, December 20). Wikipedia. https://en.wikipedia.org/wiki/Frontend_and_backend

Frontend is a what we see when we're visiting any webpage as a users. We interact with the design, click on the buttons and read the results. In this case users will even upload data to the server and pass them to 2nd layer, but they won't have any interaction with other data there (That would involve creating accounts and manipulating files or tests contents). Second layer is so called backend a layer where administrator of the page is involved. It's not necessarily the same person as the programmer, but here it is the case. Backend is a layer where internal data of the web page are manipulated so the presentation in first layer can change. Usually you add a blog post, change picture of an article or upload new media to gallery. In this case admin can add/update/delete a new charset or language to check against, admin can add a test to any filed of *name* table or just alter existing parameters of inserted tests. More complex changes in the system must be aligned with programmer. The backend will interact with filesystem by saving user uploaded font files, but also it will read data from files. These data could be definitions of test, charests, languages, metric, kerning or features. The file format for such a storage will be discussed later. Also the backend will interact with database, where all the data about each user, test or analysis will be stored.

3.3. FILE STRUCTURE

There is multiple file types used in Django natively. There are .py files – python scripts which are run on the server in backend, there are .html, .css and .js files used for frontend presentation.

But in this short chapter I would like to focus on file types which will be used by the application to store its input data for testing on the file system apart from database.

The file structure should be easy to load to the application and easy to read by human. The size of stored data does not pose a problem in this case at the size of stored data is not very big. Very easy to read by humans and even by computers is XML, so without any deeper research I would implement this format here.

3.4. DATABASE STRUCTURE

Image above shows database structure of the latest version (2025_02_13). The database structure is one of the main parts of the functionality of the Django Framework. The structure is not static from the beginning, it's growing as the functionality is added. The important decisions needs to be made straight away, because changing the structure in the later in the development would be very hard if not impossible.

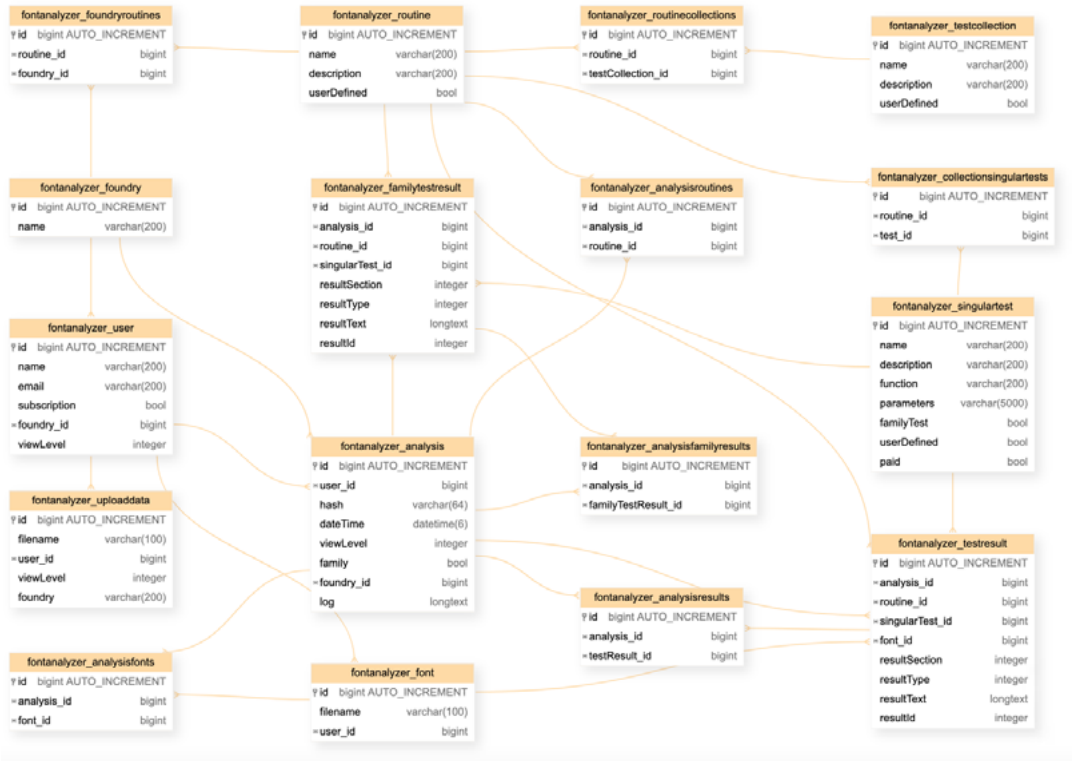
There are several main tables in the structure:

- fontanalyzer_analysis – gathers all the data about the analysis, the user, the foundry, date and time, which view level was chosen, hash and log; further to this table there are connected tables which are containing information about: analyzed fonts, routines run in this analysis, results for every test in this routine, results for every family test in family routine

- fontanalyzer_singulartest – this table contains information about every single test possible to run in the analyzer. It has name, description, function to run and parameters. This singular test is than connected to:

- fontanalyzer_collectionofsingulartests which is a set of the same kind of tests, like metadata test, language tests, etc... and these are than connected to:

- fontanalyzer_routine which describes the routine from the last table



Complete database structure of fontanalyzer.app

So to summarize the structure here – The user uploads the fonts, they are stored on the server disc storage and the analysis is created, knowing where the fonts are. Based on the level of view the user selected there are routines[6] inserted to the analysis[7] and then every test in every routine is loaded from the database and run with corresponding parameters that are either in the database or stored on the server disc as a file. The results of each test are stored in tables connected to the analysis for either singular test or family test. All of this is than read and structured on the time of rendering Frontend and presented to the user.

3.5. ANALYSIS

3.5.1. Creation of analysis

As described briefly in the previous paragraph the testing procedure starts after the user has selected view level and uploaded the fonts to the server. The font files are stored on the server disc.

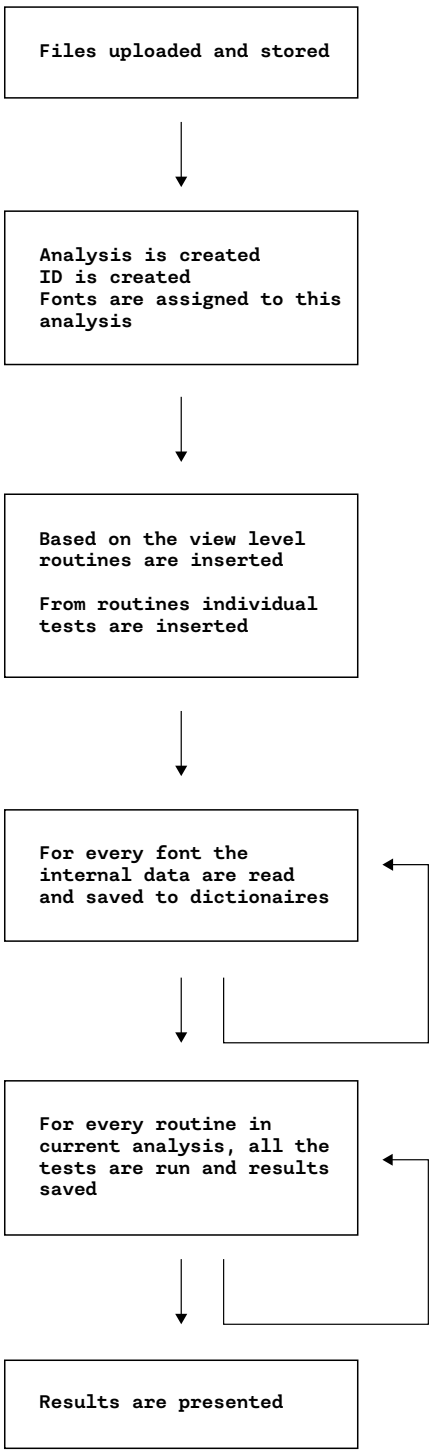
6 discussed in 4.5.1

7 discussed further also in 4.5.3

Following is creation of the analysis which holds the ID, time, view level, log or foundry (if custom is selected). All following tests and results are assigned to this analysis ID.

3.5.2. Phases (Routines)

Depending on the user level selection the routines are inserted to the process. Routine is an internal name for a group of tests (phase or better category) which will be discussed further in this chapter. There are default routines which are the same for every user level, but there are also routines that are different and are consisting of different tests or test with different parameters from the default. Also, custom tests can be inserted in the process this way. Custom test routine is bound to the foundry in the database, so user has to insert the foundry ID to run custom test. This functionality is limited only to showcase the possibility of doing so. The aim of this work is not to build comprehensive backend to this feature, but rather to focus on the testing.



Process of fonts analysis

From the book research, results of the questionnaire and my practice as FE I've come to 8 categories (routines) of tests:

- Meta
- Charset
- Languages
- Glyphs
- Metrics
- Kerning
- OT Features
- Family tests

3.5.2.1. Meta

In meta all the information about the font is read and tested. This mainly consists of reading *name*, *hhea*, *head*, OS/2 and *post* table. There are more tables containing information about specific kinds of fonts like variable fonts, color fonts, pixel fonts or AAT fonts. These fonts are minority in the production, or they are deprecated. For purpose of this work, they are not considered, but this may change later in the future. For example, support for variable and color fonts could be very useful.

"The *name table* (tag: *name*) allows you to include human-readable names for features and settings, copyright notices, font names, style names, and

other information related to your font."[8] Here's the most important information to read and test:

NAME	CONTENT
NID0	Copyright notice.
NID1	Font Family.
NID2	Font Subfamily.
NID3	Unique subfamily identification.
NID4	Full name of the font.
NID5	Version of the *name* table.
NID6	PostScript name of the font
NID7	Trademark notice.
NID8	Manufacturer name.
NID9	Designer (name of the designer of the typeface).
NID10	Description.
NID11	URL of the font vendor.
NID12	URL of the font designer.
NID13	License description.
NID14	License information URL.
NID15	Reserved.
NID16	Preferred Family.
NID17	Preferred Subfamily.
NID18	Compatible Full (MacOS only).
NID19	Sample text.
NID20–24	Defined by OpenType.
NID25	Variations PostScript Name Prefix.
NID26–255	Reserved for future expansion.
NID256–32767	Font-specific names.

[9]

"The *hhea* table contains information needed to layout fonts whose characters are written horizontally, that is, either left to right or right to left. This table contains information that is general to the font as a whole."[10] Here are most important data to read and test from this table:

8 Font Names Table – TrueType Reference Manual – Apple Developer. (2020). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6name.html

9 Font Names Table – TrueType Reference Manual – Apple Developer. (2020). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6name.html

10 Horizontal Header Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6*hhea*.html

NAME	DESCRIPTION
ascent	Distance from baseline of highest ascender
descent	Distance from baseline of lowest descender
lineGap	typographic line gap
advanceWidthMax	must be consistent with horizontal metrics
minLeftSideBearing	must be consistent with horizontal metrics
minRightSideBearing	must be consistent with horizontal metrics
xMaxExtent	Compatible Full (MacOS only).
caretSlopeRise	Sample text.
caretSlopeRun	Defined by OpenType.
caretOffset	Variations PostScript Name Prefix.
metricDataFormat	Reserved for future expansion.
numOfLongHorMetrics	Font-specific names.

11

"The *head* table contains global information about the font. It records such facts as the font version number, the creation and modification dates, revision number and basic typographic data that applies to the font as a whole. This includes a specification of the font bounding box, the direction in which the font's glyphs are most likely to be written and other information about the placement of glyphs in the em square."[12] Here's the most important information to read and test:

NAME	DESCRIPTION
fontRevision	set by font manufacturer
flags	
unitsPerEm	range from 64 to 16384
created	international date
modified	international date
xMin	for all glyph bounding boxes
yMin	for all glyph bounding boxes
xMax	for all glyph bounding boxes
yMax	for all glyph bounding boxes
macStyle	Bold, italic, underline...
lowestRecPPEM	smallest readable size in pixels
fontDirectionHint	Mixed directional glyphs

13

11 Font Names Table – TrueType Reference Manual – Apple Developer. (2020). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6name.html

12 Font Header Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6head.html

13 Font Header Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6head.html

"The OS/2 table consists of a set of metrics that are required by Windows. It is not fully used by Apple platforms."[14] Here's the most important information to read and test:

NAME	DESCRIPTION
version	table version number (set to 0)
xAvgCharWidth	average weighted advance width of lower case letters and space
usWeightClass	visual weight (degree of blackness or thickness) of stroke in glyphs
usWidthClass	relative change from the normal aspect ratio (width to height ratio) as specified by a font designer for the glyphs in the font
fsType	characteristics and properties of this font (set undefined bits to zero)
ySubscriptXSize	ySubscriptXSize;
ySubscriptYSize	recommended vertical size in pixels for subscripts
ySubscriptXOffset	recommended horizontal offset for subscripts
ySubscriptYOffset	recommended vertical offset form the baseline for subscripts
ySuperscriptXSize	recommended horizontal size in pixels for superscripts
ySuperscriptYSize	recommended vertical size in pixels for superscripts
ySuperscriptXOffset	recommended horizontal offset for superscripts
ySuperscriptYOffset	recommended vertical offset from the baseline for superscripts
yStrikeoutSize	width of the strikeout stroke
yStrikeoutPosition	position of the strikeout stroke relative to the baseline
sFamilyClass	classification of font-family design.
panose	10 byte series of number used to describe the visual characteristics of a given typeface
ulUnicodeRange	Field is split into two bit fields of 96 and 36 bits each. The low 96 bits are used to specify the Unicode blocks encompassed by the font file. The high 32 bits are used to specify the character or script sets covered by the font file. Bit assignments are pending. Set to 0
achVendID	four character identifier for the font vendor
fsSelection	2-byte bit field containing information concerning the nature of the font patterns
fsFirstCharIndex	The minimum Unicode index in this font.
fsLastCharIndex	The maximum Unicode index in this font.

14 OS/2 Compatibility Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6OS2.html

sTypoAscender	The typographic ascender for this font. This is not necessarily the same as the ascender value in the *hhea* table.
sTypoDescender	The typographic descender for this font. This is not necessarily the same as the descender value in the *hhea* table.
sTypoLineGap	The typographic line gap for this font. This is not necessarily the same as the line gap value in the *hhea* table.
usWinAscent	The ascender metric for Windows. usWinAscent is computed as the yMax for all characters in the Windows ANSI character set.
usWinDescent	The descender metric for Windows. usWinDescent is computed as the -yMin for all characters in the Windows ANSI character set.
ulCodePageRange1	Bits 0–31
ulCodePageRange2	Bits 32–63
sxHeight	The distance between the baseline and the approximate height of non-ascending lowercase letters measured in FUnits.
sCapHeight	The distance between the baseline and the approximate height of uppercase letters measured in FUnits.

15

"The *post* table contains information needed to use a TrueType font on a PostScript printer. It contains the data needed for the FontInfo dictionary entry as well as the PostScript names for all of the glyphs in the font. It also contains memory usage information needed by the PostScript driver for memory management."[16]

NAME	DESCRIPTION
italicAngle	Italic angle in degrees
underlinePosition	Underline position
underlineThickness	Underline thickness
isFixedPitch	Font is monospaced; set to 1 if the font is monospaced and 0 otherwise (N.B., to maintain compatibility with older versions of the TrueType spec, accept any non-zero value as meaning that the font is monospaced)

17

15 OS/2 Compatibility Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6OS2.html

16 Glyph Name and PostScript Font Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6post.html

17 Glyph Name and PostScript Font Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6post.html

All these table are vital part of testing metadata. Most of these values can be tested against computed values from other data in the font or it can be tested against definitions OT Specification (specified maximum length, specified logic, exact values)

3.5.2.2. Charset

In this stage the font is tested in comparison to specified charset. The table *cmap* contains information on what characters are included in the font. The stored information is character code (for example Unicode) and it's mapping to the glyph ID. One glyph can have multiple codes assigned. Example of this is uni-case font where uppercase and lowercase share the same glyph ID for each of the cases of a letter.

There are many definitions of charsets from the past. As example we can take Adobe Latin 1, 2, 3, 4, ASCII, ISO-8859-1, Windows 1250–1258. They all come from the times before Unicode where only 256 glyphs were allowed in one font file and therefore the file had to be interpreted to be read in the correct way for the operating system and selected language. Nowadays there are modern charsets like Opentype Latin Pro or Latin Extended – A or B. There are also many customized charsets of type foundries. The aim of this work is to prepare testing for any charset and to gather the most basic ones and test against them. New charsets will be added, and it must be easy and straightforward work. As mentioned before XML will be used as a format and it will be stored on the system disc as a file. Then it can be added, modified or deleted easily.

The overall list of glyphs and Unicodes can be presented to the user.

3.5.2.3. Languages

This stage seems at first very similar to the previous, because it is in principle the same – testing against defined charset. I've decided to split these two to prevent mixture of the terms and get more visual clarity to the results. One thing is to test against charset – that is something very type foundry specific and another thing is testing against language which is more client/user specific. Designers often promote fonts with number of supported languages.

For purpose of this work, I will gather most of the European Latin charsets and process them again into XML definition files. These will be then tested against and presented. Quality source data for compiling the language definitions can be found in Hyperglot made by Rosetta Type.

3.5.2.4. Glyphs

Glyphs stage should show the shape and details of every glyph in the font. For such a task the application has to read the outline data and present them to the user. There are two kinds of outline data in current format of the fonts:

- OTFs are using postscript outlines – counterclockwise cubic Bézier curves (3rd order). Such an outline is defined with 2 points and 2 handles in case of curved line or just two points in case of straight line. FontTools library I use to handle the font data can convert such an outline to SVG commands and these can be rendered on the results HTML page.

- TTF outlines are made of clockwise straight-line segments and quadratic Bézier curves. These curves are mathematically simpler and faster to process than cubic Bézier curves, which are used both in the PostScript-centered world of graphic design and in Type 1 fonts. However, most shapes require more points to describe with quadratic curves than cubics. This outline can also be converted to SVG by FontTools library.

Apart from displaying the outline, which does not give user any added information about the outline there are few values that can be presented alongside:

- Unicode
- Width
- Left sidebearing
- Right sidebearing
- Xmin, Xmax
- Ymin, Ymax

These values can be read from *glyf/cff* table and from *hmtx* table.

From my research in tools and questionnaire there are few tests that can be performed on the outlines to assure the needed quality – for example test for unevenness or smoothness of curved outline. Another parameter that can be tested is wether the points are in extremes. While not strictly required by font specifications, ensuring points at extremes improves interpolation and hinting reliability.

3.5.2.5. Metrics

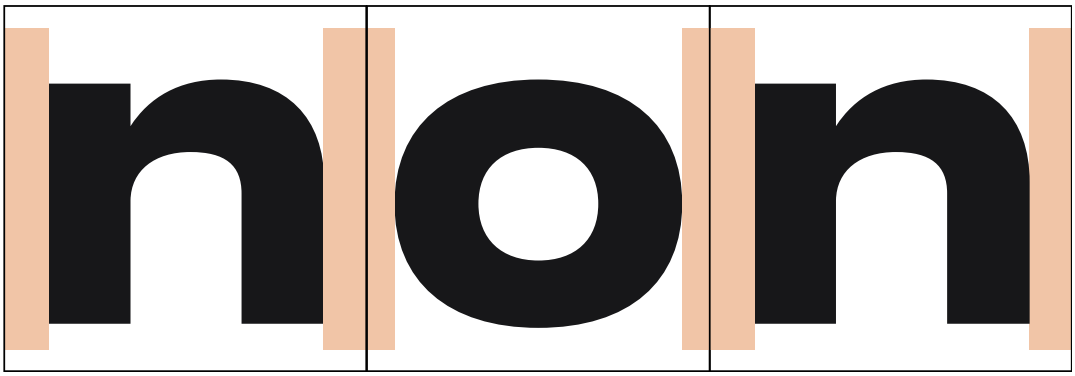


Figure is describing metrics on each side of the glyph

Metrics are information about overall whitespace in the font. Every glyph in the font is having default space on both sides of the design – right and left sidebearing.

In stage of metrics the data are read from *hmtx* table and the table where outlines are stored. The data are consisting of left sidebearing, right sidebearing and width for every glyph. This data can be presented to the user and also tested. The test can be for example if empty glyphs are having 0 in sidebearings or if comb glyphs are having 0 in width. Also, glyphs with the same design should have the same metrics.

In many tools to test font there is a test text field consisting of sample metric texts to see if the font performs well. This can be also included in the presentation to the user.

3.5.2.6. Kerning

Figure is describing kerning between few pairs

Kerning can be considered as exceptions to metrics to get overall pleasant whitespace distribution. There are two kinds of kerning:

- flat kerning
- class kerning

Flat kerning is positioning one left glyph to right glyph by value. Class kerning positions one class to another class. The class stands here for the representative of the same design on the left or right side of the glyph. This is very useful when working with accented letters, where usually the diacritic does not interfere with whitespace on any side (there are exceptions of course) Both in flat and class kerning you can position the same glyphs together. Class kerning saves a lot of superfluous data that had to be otherwise saved as flat kerning. Here's one visual example of kerning principle between glyphs that otherwise would create large whitespace gaps between shapes.

Kerning data are read in this stage from GPOS table. Older fonts have their kerning stored in *kern* table. For the scope of this work this table will be omitted and no kerning will be read from this deprecated table. All modern fonts are having GPOS table.

Testing kerning is quite a difficult task. There is no 'right' value of any kerning pair that can be tested out of the box. Even if some design principle is being considered good practice and that would suggest one way of kerning, there may be another way of designing the glyph and another way of kerning and we must consider as possibility. One example of this is dcaron. Today's good practice is to design dcaron with negative right sidebearing and thus using positive kerning for forthcoming glyphs with ascender or cap size stem on the left side. (ď'l, ď'k, ď'!, ...) Although positive kerning for these pairs is a best practice, alternative designs might justify different kerning approaches. Rather than enforcing fixed kerning values, the tool identifies inconsistencies – like sudden jumps in kerning values across masters – that could indicate design errors or incorrect interpolation.

Here comes the custom level of testing to play. For certain foundries/schools/institutions testing certain font families there could be minimal kerning list of kerning pairs to be included in the font. There could be certain requested way of designing glyphs and thus implicating certain values in the kerning pair and this can be tested. But only in custom level and only when user will define it in the custom testing.

Kerning also can be displayed to the user. Such a list can be quite extensive and hard to read, so there must be some kind of filtering to make it more usable to read.

3.5.2.7. OT Features

Figure is describing opentype ligature substitution

OT features or advanced typographic “layout” features which prescribe positioning and replacement of rendered glyphs. Replacement features include ligatures; positioning features include kerning, mark placement, and baseline specification.

It brings improvements in rendering combinations of glyphs by selecting more appropriate version of the glyph, it can substitute few glyphs to one joined glyph (ligature) and for many non-Latin scripts this functionality brings the only solution to shape complex combinations of glyphs that requires joining, accenting or reshaping some or all the glyphs. It can also decompose previously composed glyphs (in case of higher values of tracking for example)

OT Features are read from GSUB table. Similarly, as kerning there is no required opentype feature that has to be inserted to every produced font.

Again, here the custom users could define their own minimal set of OT Features or content of one specified feature like calt.

3.5.2.8. Family testing

All the 7 stages mentioned before are performing single font testing without taking other fonts in the family into account. 8th stage – Family is here to make sure that the font works not only as single font but as a part of a font family. Font family consists of more fonts, which are usually reflecting on the design an ordered sequence of values on a design axis. This design axis can be anything that can change the design or the rendering of the fonts. Most known axes are – weight, width, italic/slant angle, optical size, grade.

In the family testing stage, all of the previous stages can be tested in order to compare the values in the whole family.

- In Meta stage the family should be consistent in naming, usWeight, usWidth and slanting indicators. Many of the information values should be the same as copyright, trademark, version, designer, manufacturer and so on. This may seem unnecessary as usually the whole family is exported in one run from one software, still there is a use case where for example italics are stored in another project file and the metadata could be different by accident. This may result in error message.

- In Charset and Language stage all the fonts usually should meet same charset and language support, but there can be cases as true Italics where there can be alternates of the upright design as stylistic set and therefore having different charset as the upright version. Therefore, this may result only in warning message.

- In Glyphs stage user may want to visually compare the design progress of the whole family in each glyph. This helps prevent incorrect interpolation between two masters. Also, in this stage every glyph can be tested to have the same number of nodes and components. This also may point to incorrect interpolation.

- In Metrics stage again all the metrics (left sidebearing, right sidebearing and width) of each glyph of the family can be compared by value, but that does not really give meaningful information to the user. There are simply too many numbers. What can be done (it was pointed out in one of the testing I've conducted with real FE) is to find glyphs that are deviating from the average delta of one metrics to another style. For example, there can be average delta 10 points from Light to Regular style and if there is a glyph that is having delta more than 50 points it is very suspicious. This can lead to forgotten/strange metrics values or to improper interpolation where the glyph design is collapsing and changing the value of one or more metrics abruptly.

- Kerning stage could show progress of kerning in each pair, but again as said before, the testing is ambiguous here. There can be huge difference between values from pair to pair. So, the same principle as in metrics cannot be applied here. Also, the progress in one pair may not be linear (the cause can be for example volatile metrics) and this may prevent to take conclusions about any value. There could be filtering present to help user navigate through the values.

- OT features stage should test the same number of features and glyphs in the features. This also may vary in the family so it may result only in warning message.

3.5.3. Results

Every test has a result. It's either error or a warning or combination of both. The application has to store the results for every test performed in a way that it can be presented to the user in a meaningful way. First level of filtering the output is the above-mentioned test division into routines. This sorts the results for the user into groups that are clearly labeled and understandable.

3.5.3.1. Assigning errors and warnings to font data

For every routine there is a content and tests. The content is there to inform the user about the state of things in the font file. Not showing any signs of errors or warnings, just plainly informing about the data there. To this layer of information there has to be some way to include the results of all the tests run on these data. I've decided to give every read value from the font an unique ID that could be then referenced in the error or warning and displayed with it. In the end of one test there can be none, one or multiple errors or warnings assigned to one ID. Also multiple tests can reference the same IDs and effectively adding results to some kind of list. The IDs will be assigned to values while reading the font file and there are unique to the whole analysis (and in the range of all the fonts). The IDs are not interchangeable between any fonts, every font has different IDs depending on contents that are read. That also implies that when analysis is completed the stored values are relevant only to the font files assigned to the analysis and the result cannot be interpreted without these. The IDs without the fonts has no meaning, only this app can reconstruct the results from these IDs.

Also this brings the question of reviewing old results in newer version of the app. If the app is updated with more tests that are inserted in between already existing ones, the ID sequence will be broken, and the results will fail to render correctly. This must be evaluated if that poses a significant problem to the users. In my use cases I consider such a testing as valid only in very short-term time frame.

3.5.3.2. Structure of the result data

The information saved in a result must inform the user about the error or warning in a meaningful way. By storing only ID of the data where the issue occurs, we are losing context. To add the context there must be a message stored in addition of the ID. To categorize the results furthermore I've came to conclusion that the app can have a storage where all the common knowledge about such an issue can be stored (links to the specs, use cases, more detailed descriptions or rationales). This would need another ID, but this time relevant to the issue and these IDs would be consistent through all the tests and analysis. It will be like a reference to a list of errors and warnings. Such a list or database of errors and warnings and reasonings to them would be very time demanding and for purpose of this work I'm only building the structure to enable it, but I'm not filling it up with data.

This structure comes with one restriction that implies the message to the user is text only. When researching all the tools mentioned in the questionnaire and around the Internet and development applications, I found that it is hard to decode some written results regarding design or outline. Reading for example that line 403, 211 to 401, 576 may be uneven is hard to visualize even for trained font engineer. Having tens of such a messages to verify can be very time consuming. I've decided to include rendering for such a cases to the SVG outline in glyphs test category. If this app should reach high standards of usability, this must be done.

Lastly each test should be able to save different results for different view levels. Beginners cannot be confronted with very specific font engineering language and overwhelmed with numbers and tables that are not meaningful to them. To meet such a requirement every test must store results corresponding to each of the 3 view levels (custom is using one of the 3 view levels it is not a different view).

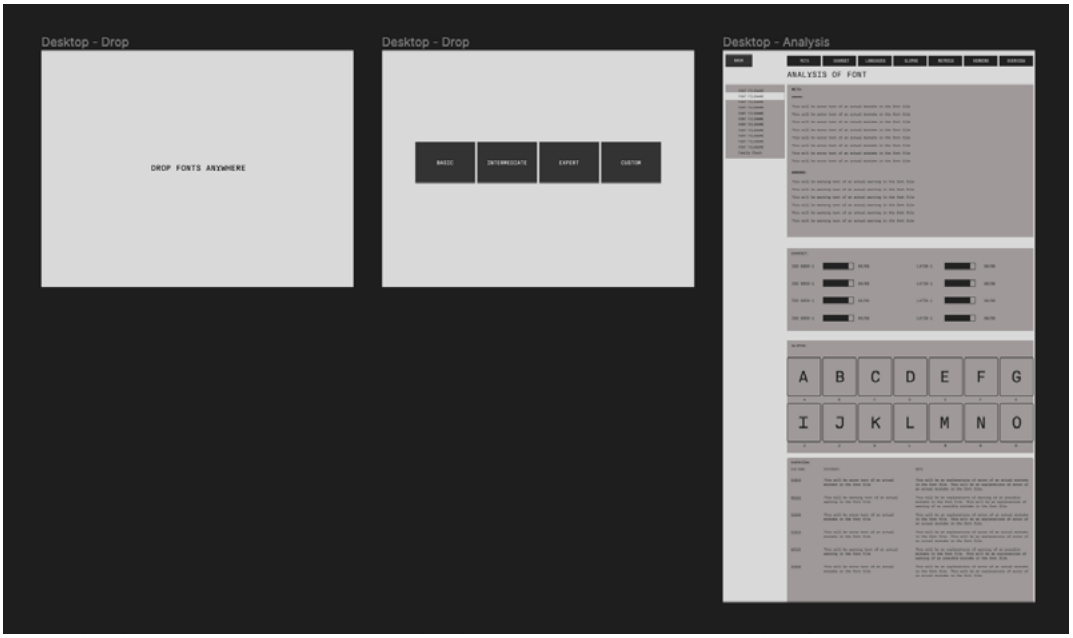
4. DESIGN

As discussed before process of composition of content and visual elements (designing) must be done with regard to user centered design. UX design process begins at understanding the objectives of a business and how best to serve a target audience.
This mean in this case to research existing solutions, identify the objectives of this app and prepare solution that will be used with ease. Good start is to plan what should be shown on which page of the application and build low-fidelity representation of the layout. In the beginning I've listed 4 steps where the first one happens on the side of the user:

1. preparation of material to be checked (on the side of the user)
2. selecting appropriate check kind/level
3. inserting this material to application/form
4. receiving results

4.1. ONE ENDLESS PAGE

In web environment the user comes to the initial page, selects the appropriate check, inserts the fonts to the webform and receives the results. So basically there could be 2 or 3 steps. My initial sketch of these steps and their lo-fi presentation was this:



First wireframe of the app

The design consists of 3 steps out of 4 as mentioned before, where the first step is left on the user side. In the 2nd step the user will drop fonts to test to page by dragging them from folder in the computer. In the 3rd step the user will select desired test level and in the 4th step user will receive the results.

Immediately after testing this first sketch, I've realized that step 2 and step 3 can be merged into one by letting the user select which level of check is desired by dropping the fonts on one of the 4 buttons in the page 3. This makes the process faster and eliminates one reading and clicking action of the user.

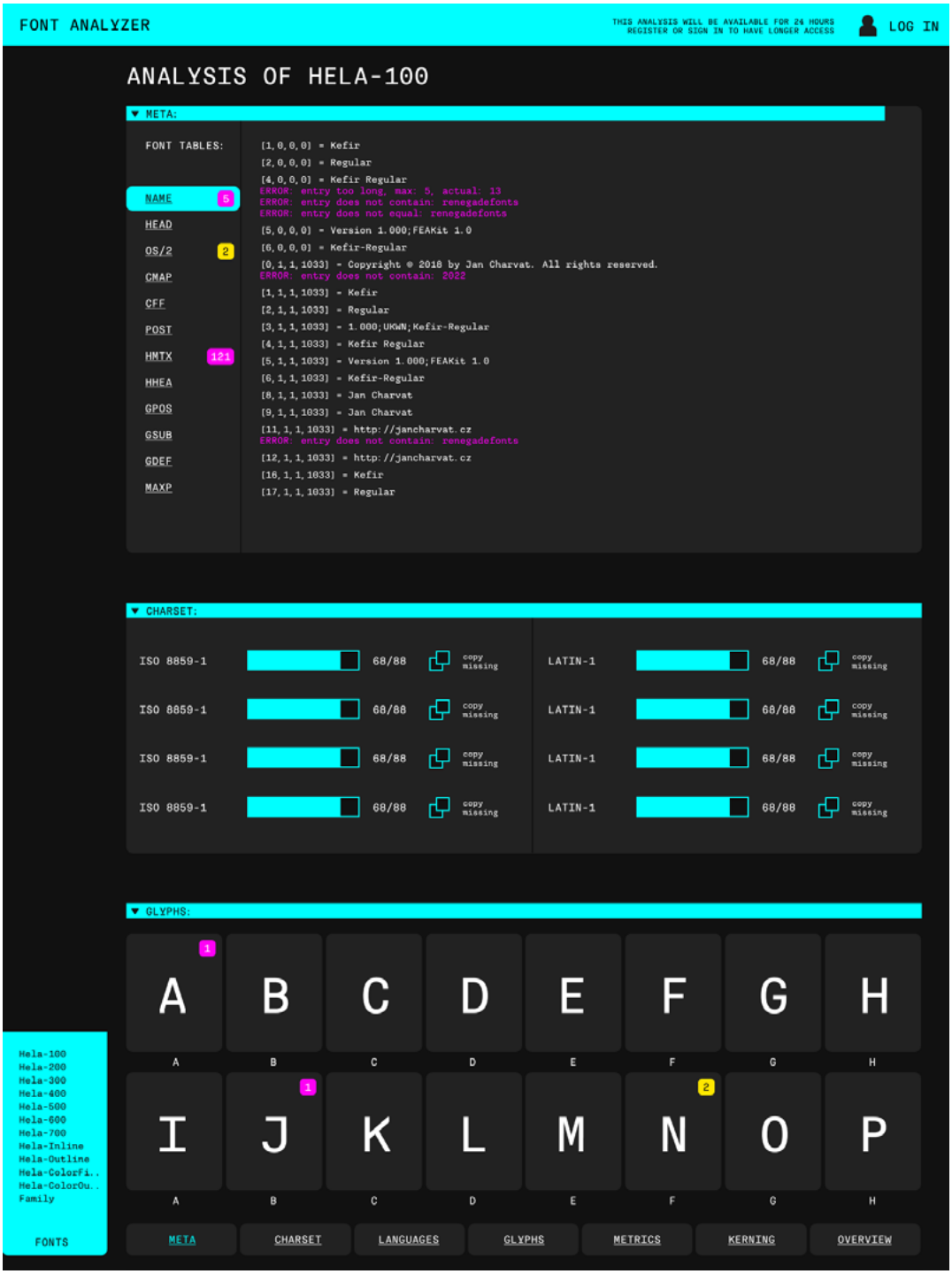
The 4th step is presentation of the results to the user. My first wireframe was consisting of 3 main parts. First is the font menu, where user can select and click any font to change the results to show results of the corresponding font or user can select family testing and the family section will be shown. Next part is the selection of what routine will be shown in the window frame. There are all of the 7 routines with 8th being the overview of all the errors and warnings. The menu was initially placed on top of the window not to interfere with the content. Lastly most of the window is taken by presentation of the results. In first wireframe it took shape a one long single page with sub headlines to divide the routines from each other.

Later in the process I've added colors to the design and more detailed wireframe of the results page. I've decided to start with these 5 basic colors:

#111111 (background)	#EEEEEE (text)	#40E0D0 (highlight)	#FF00FF (error)	#FFFF00 (warning)

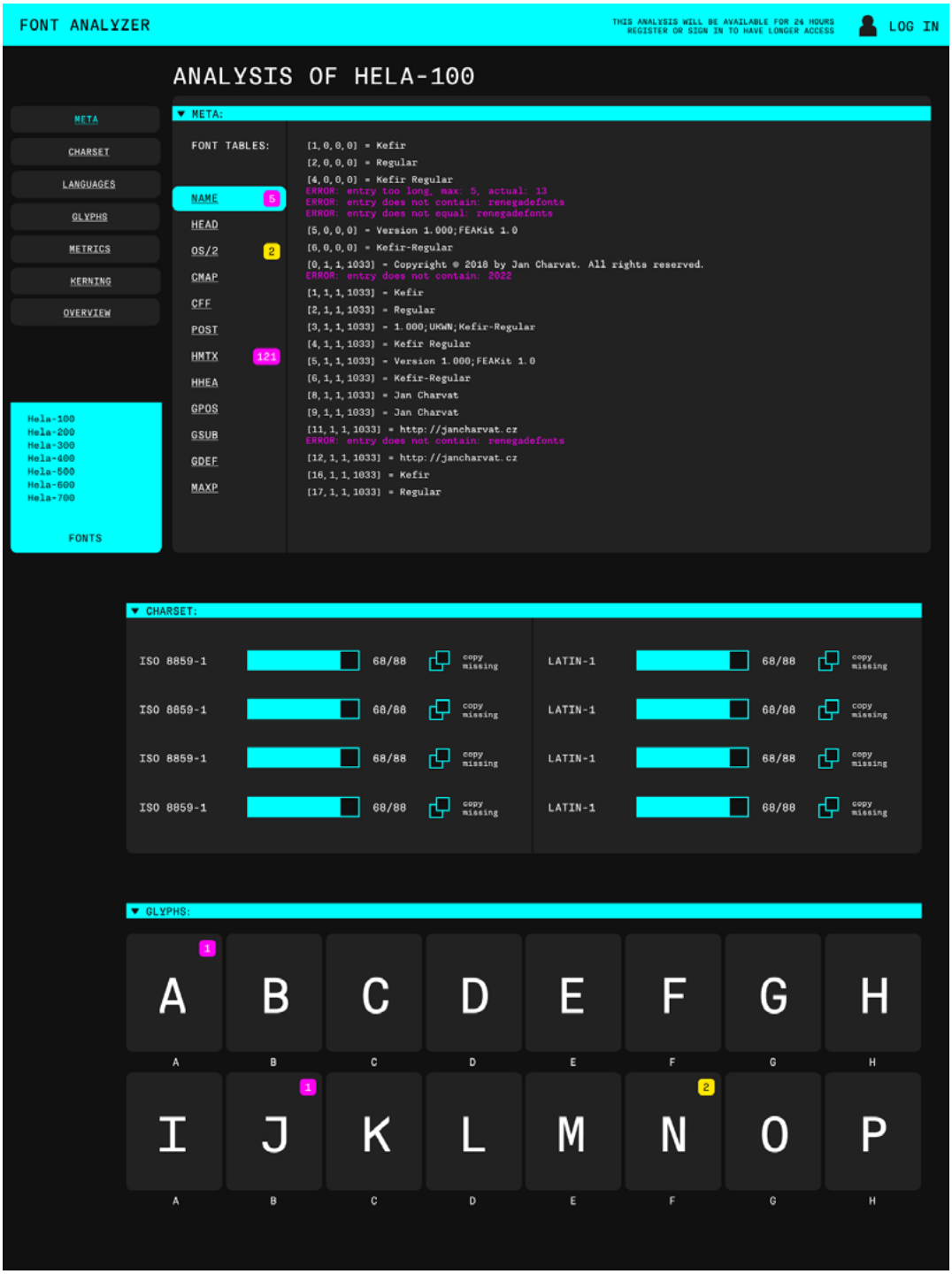
Color palette of the first designs

The idea behind this color scheme is CGA monitors palette. This color scheme should represent the precision and power of the console programs. Background and text color are having big contrast to make text as much legible as possible. Highlight color is for headlines, selection or graphs. With error color it will highlight wrong values and error text and the same goes for yellow and warnings. This color scheme is now used in *dark* mode of the web application.



First sketch of results presented on one long page

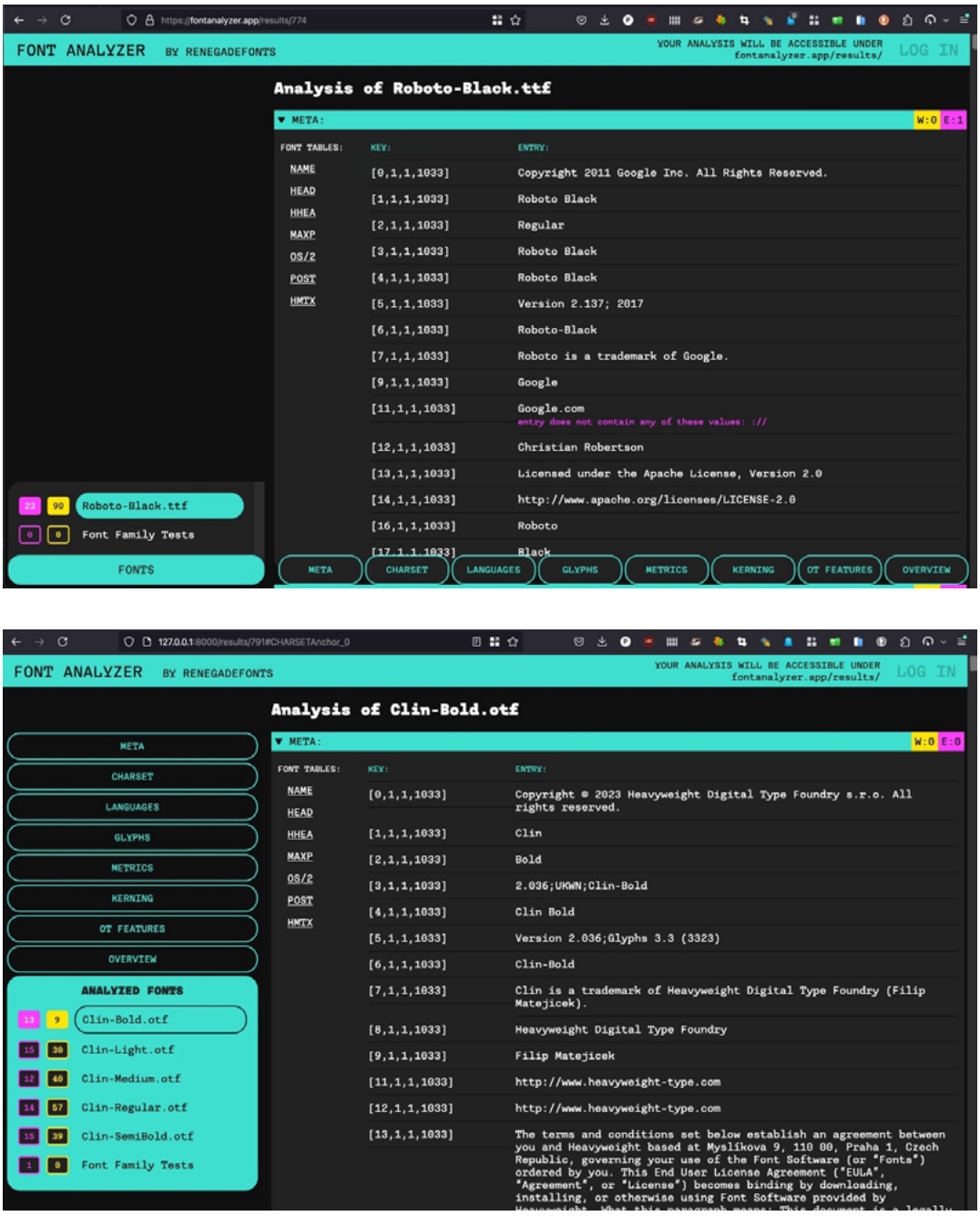
This more detailed wireframe shows first draft of design of the results page in color. First in the results is the display of meta data read from the *name* table. This part is divided into two parts where the left part allows the user to select one table and display its contents on the right side. There are error messages and error count shown there. Next routine shows charset fulfillment in bar charts and possibility to copy the missing ones with one click of a button. The last routine is a display of glyph set and the indication of errors and warning the some of the glyphs. In this version the routine menu moved to the bottom side. Still quite underdeveloped part was the sidebar with list of fonts. There is no indication of which font is selected and no indication of how many errors and warnings are there.



Second sketch of results presented on one long page

In next iteration I've tried to move the bottom menu to the sidebar where there was a bit of negative space that has no usage. This way the sidebar has to be slightly wider and the font list too. The font list was still undeveloped, but this moved initiated the idea of showing more information there.

These wireframes made it to testing in the local development instance of the server. Here are two screenshots of such a rendering.



Third and fourth sketch of results presented on one long page

Here you can also see the design progress of in the font list where it shows the number of errors and warnings. The background color of the font list was also tested in this run. The dark rendition was chosen as less distracting and more readable. The colored numbers on colored background were impossible to recognize or read at all. I've tried all the possible combinations of text and background colors, but none of them worked.

The position of the routine's menu was tested and here I realized that having 8 routines together with 6 or more fonts in one column creates quite a big information cluster which is hard to navigate in. The positive background helped to distinguish the font list from the routine menu, but the overall strong color brightness brought too much attention to place which does not require it.

The conclusion from this was to make the font list with dark background and to keep the menu stick to the bottom of the page.

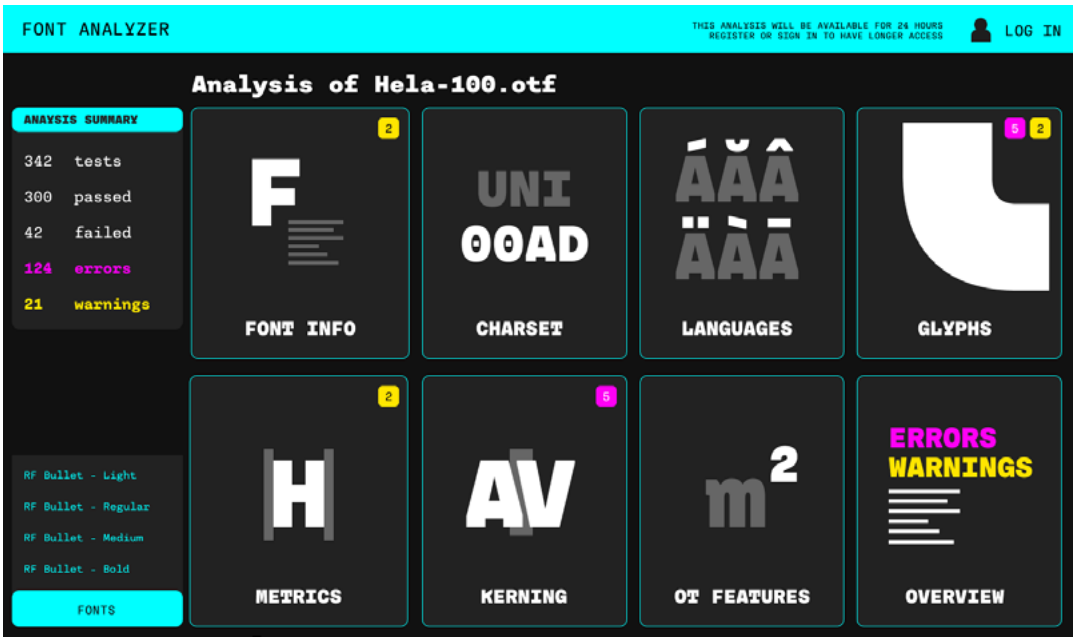
This rendering stayed for longer development time, and it was first presented as a functional web page to few font engineers for testing. The result of the testing was that the rendering was a bit overwhelming even for experienced font engineers. I've also shown this to few type designers, and they got lost in all the information. This rendering was also presented in the mid-term of my study on this university and the review of my opponent mentioned that this needs to be improved because this tool must be easy to use for non-experienced users too.

4.1.1. Tablet like routines

In response to these reviews I've came with different layout that would divide the routines into separate pages. The main page would function as a signpost to help user direct through the results. Here's the first wireframe that also accompanies the statistics of the analysis.

First tablet sketch of the web application

This was followed with more detailed rendering:

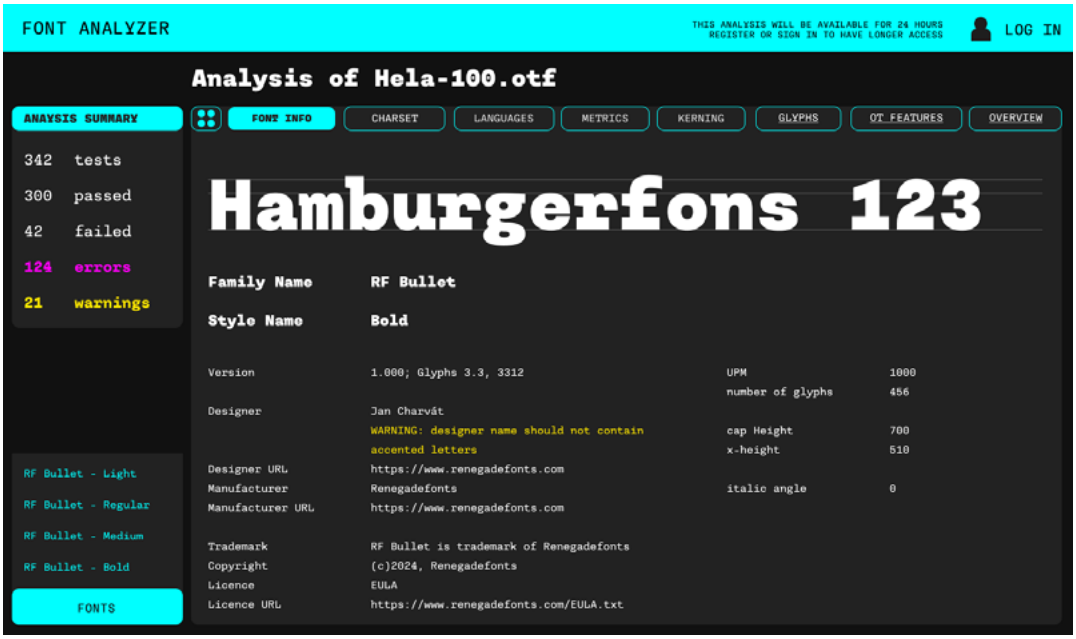


More detailed tablet sketch of the web application

Here there are icons for every routine and there are notification like numbers in some corners of the routine buttons. The statistics were moved to the left top part and this place finally found its content which goes well with the font list in the bottom.

The overall page length of the first design was getting way too high. Wireframing shows quick structural solutions on small exemplar data, but when you implement the solution and load a font with 1000 glyphs the page gets excessively long. The user may get lost and frustrated. The decision was taken to cut the long page into 8 pieces and each subpage consists of results relevant only for the one routine.

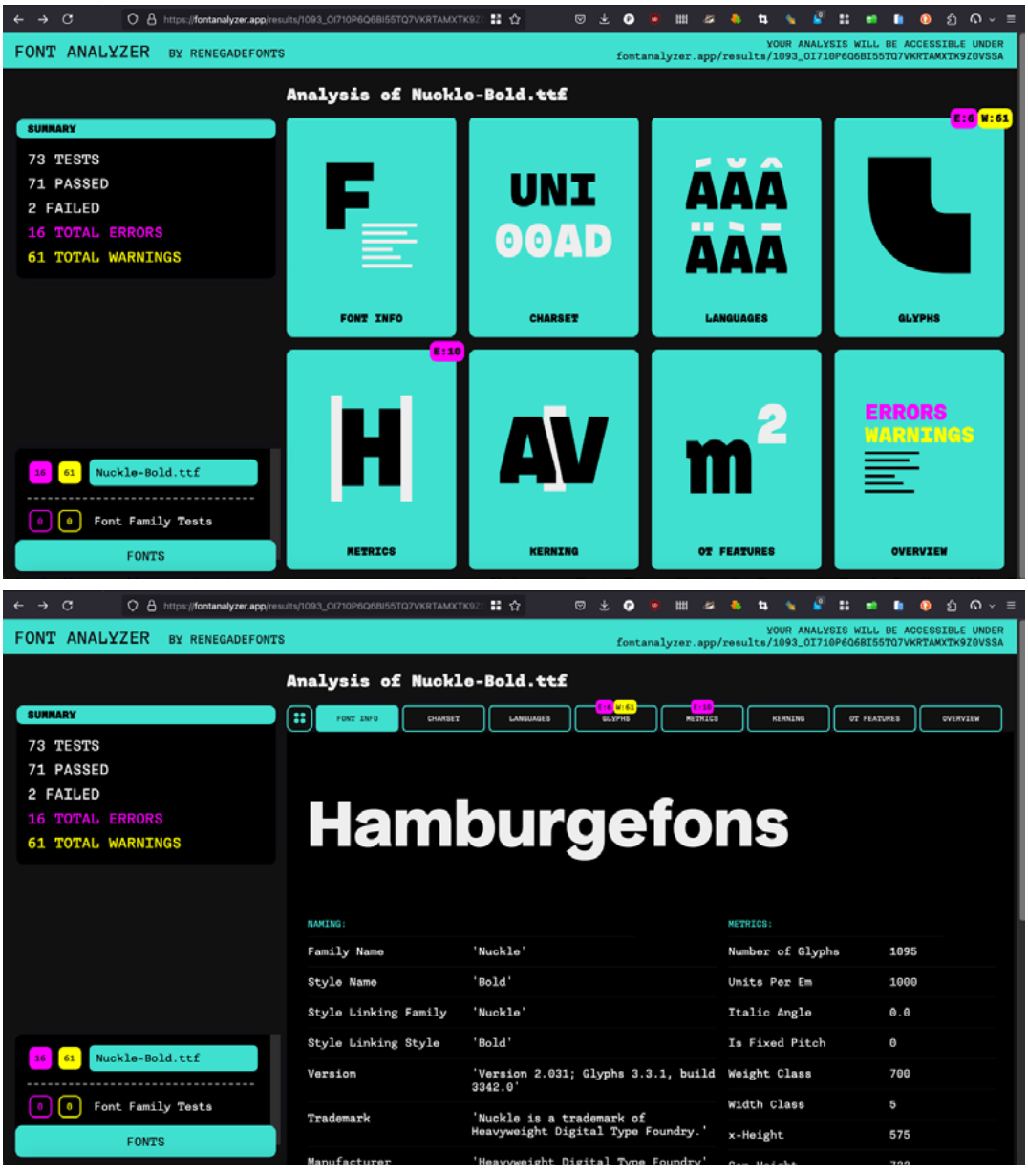
Here is detailed rendition of meta routine results:



Sketch presenting results of Meta routine

You can see that in comparison to the one page design this page is focusing on better data organization, removal of unnecessary and ambiguous entries. It highlights the most important data and works better with negative space. This page is more inviting to the user than the plain list of entries from the font table.

Both changes were implemented to the code and resulted in this rendition:

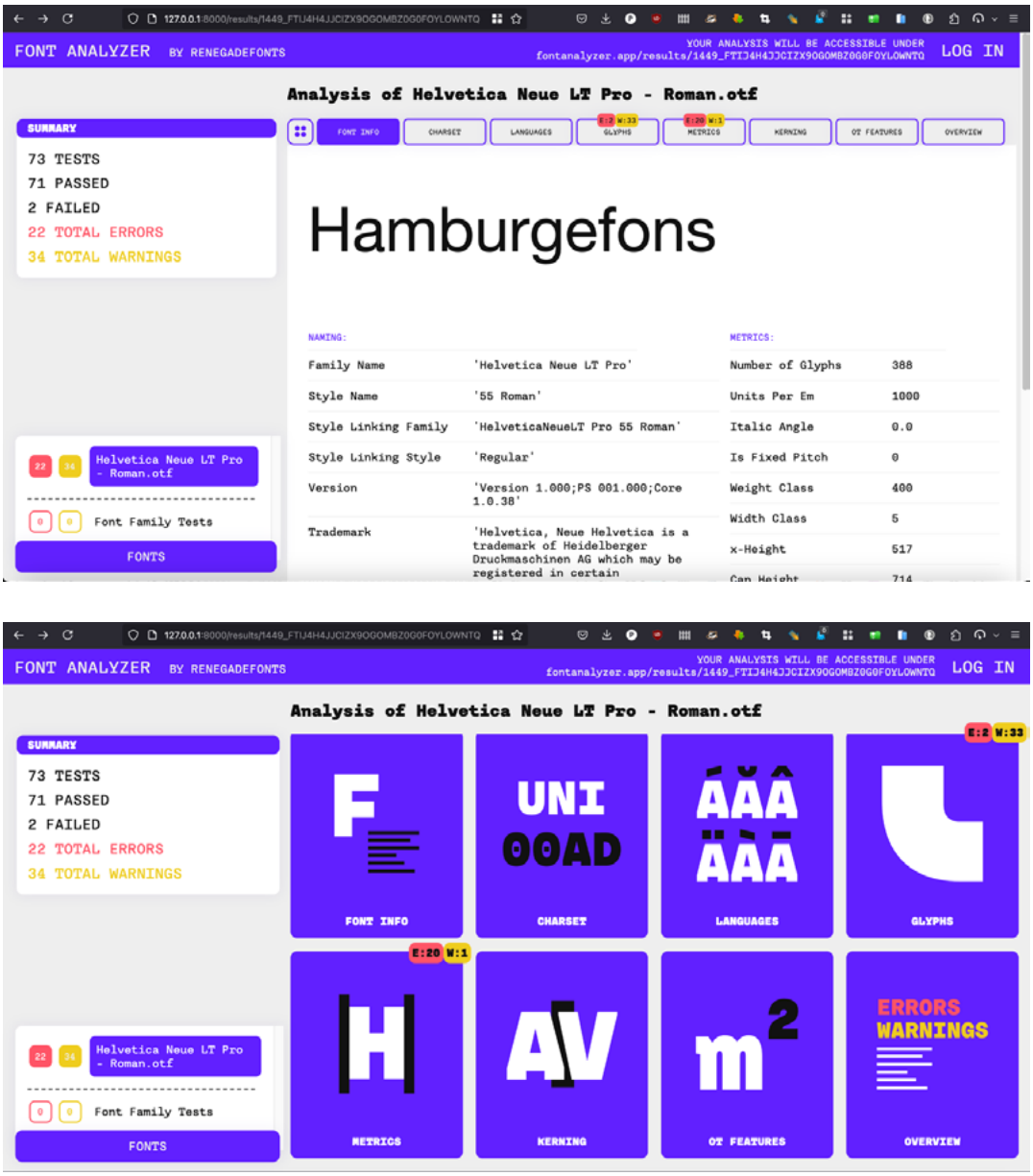


First implementation of tablet view of the web application

This was again tested, and the response was much better than with the first one-page design. From these testing users were delighted with the organization of the information, but some of them commented on the dark mode like color scheme. Some mentioned that is looks to console like and too geeky. I was resisting the need for a long time, but for the usability's sake I came with white color scheme too.

#EEEEE (background)	#111111 (text)	#6020FF (highlight)	#FF00FF (error)	#FFFF00 (warning)

The combination is not a pure negative to the dark scheme, but choses the violet as a main highlight color. The choice is intentional as red and yellow is reserved for errors and warnings, green is used in the dark scheme and blue and violet was the only remaining possibilities. I've chosen violet. The result of using such a color scheme is here:



Second color palette of fontanalyzer.app

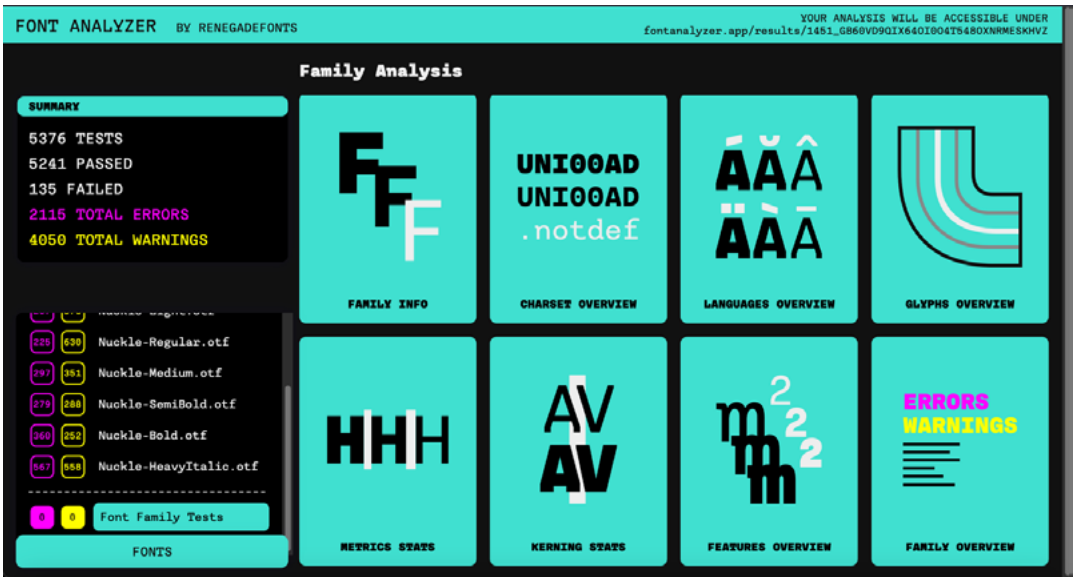
This is the design I've made progress to in this moment. The webpage is rarely in final form and the design will evolve in the future to accommodate new functionality and results of new tests. The responses to current visual style and usability are mostly positive and to quote one of the testers "it is pleasant to use" was his immediate message.

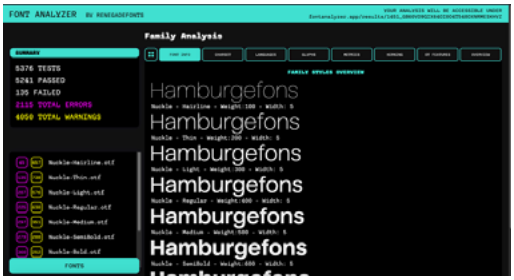
4.1.2. Design of each routine subpage

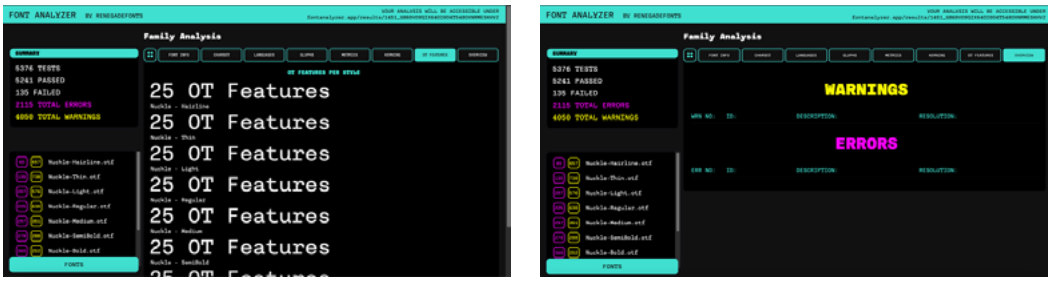
Implementation of all the routines results of single font

For the all the charts rendering there is open-source library chart.js used.

4.1.3. Design of each family main page and each family routine subpage

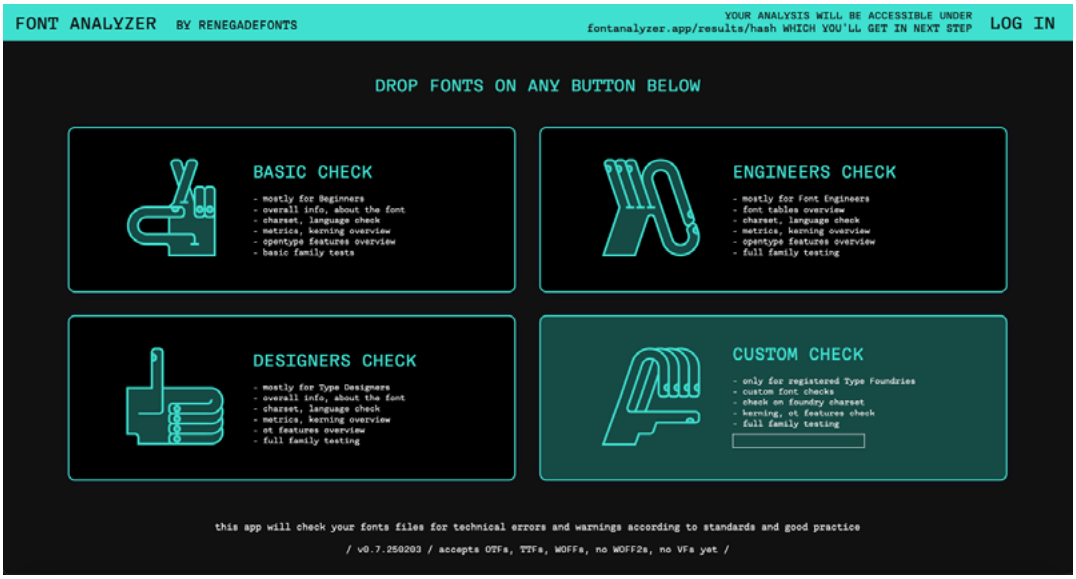






Implementation of all the routines results of family testing

4.1.4. Design of homepage / upload page



Implementation of the homepage

The homepage was divided into 4 parts. Each view level got its own illustration[18] and description to help user distinguish between them. Custom level has text input field on top of the others for users to be able to insert their foundry name. This name will be compared to stored foundries and if there is a match and corresponding routine, the user will have custom tests in the result page included.

4.2. FONT

Font analyzer font - RF Bullet - Light

Font analyzer font - RF Bullet - Regular

Font analyzer font - RF Bullet - Medium

Font analyzer font - RF Bullet - SemiBold

Font analyzer font - RF Bullet - Bold

18 Author of the illustrations is my colleague Tereza Turková and I have all the rights to use it.

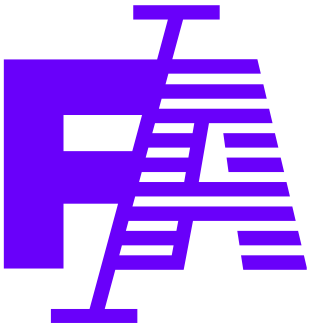
For the whole user interface of this application there is a font family RF-Bullet deployed. The font is monospaced which makes many design aligning tasks easier and gives user some kind of technical precision feel.

I'm the only designer of the whole font family and I have all the rights to use this font in any application, any territory, any time and in any amount.

4.3. VISUAL STYLE

The result of this thesis is a working online-tool. It is a software with all the needs software has. As a part of this thesis, I've also designed a logo and created social media account where I will inform public about future version updates and new tests being inserted. The account can be also used to inform broader audience about basics of font engineering.

4.3.1. Logo



The idea behind this symbol is that F and A are the first letters on the application name. The line there is like a scanning beam going through the symbol and A is made of lines because the analyzer is beaming through the structure of the letter.

4.3.2. Logotype

Vertical version

FONT
ANALYZER

Horizontal version

4.3.3. Font

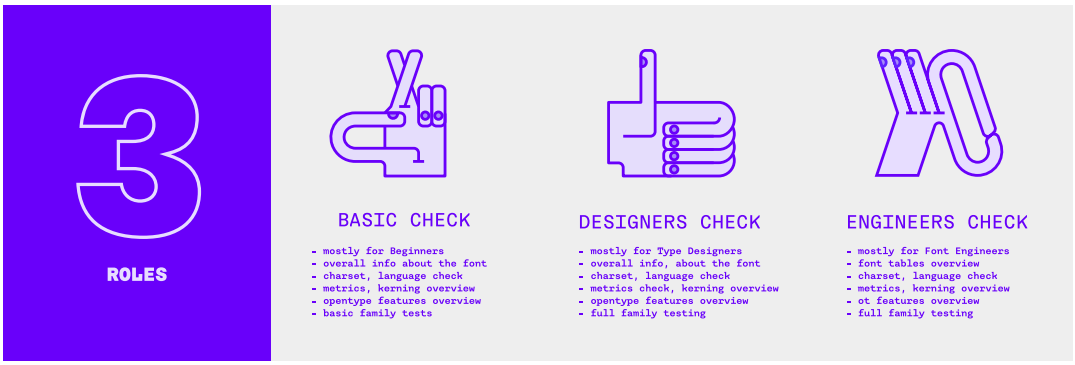
Font used is the same font used in the application.

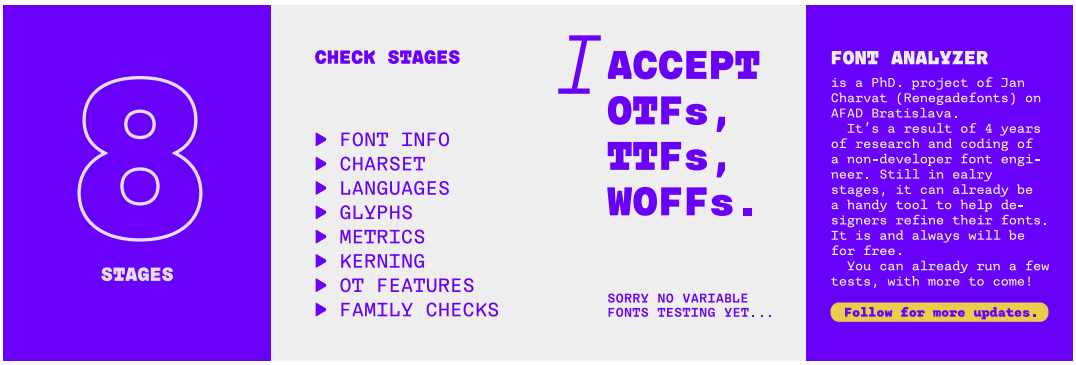
4.3.4. Colors

Colors used are the same colors as in the application.

4.4. INSTAGRAM ACCOUNT

I've created an Instagram account with name @font.analyzer and here's the design of first post. It's consisting of 11 slides. This post is informing the audience about the existence of the font analyzer application and it's features.





11 screens of first font.analyzer post

There is also a 2nd post which is showing the audience the way it is meant to be used:

11 *screens of second font.analyzer post*

I have a 3rd post proposal, which could educate the audience on basic topics:

proposal of educational post

5. FONT QUALITY

Let's define quality font in today's terms according to the technical level of type design and tools used to produce font files. In the broadest view we look on the font from the design point of view and the technical point of view. I would like to briefly describe both sides of the quality criteria here, but for the purpose of this thesis only the technical criteria will be considered relevant. There may be an algorithm or artificial intelligence instance to assess on the design quality of a font. But to my knowledge there is none today and this work has no aims to do so.

5.1. WHAT IS QUALITY?

The ISO definition for quality is "conformance to requirements- the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs." [19] , Philip B. Crosby has defined quality as "quality conforming to specifications.", M. Juran has described quality as "quality is fitness for use." and Deming has described quality as "quality is customer satisfaction."

We can conclude from above that quality product conforms to some standards and is usable in a way that user is satisfied. In font industry there is a well-maintained up-to-date standard described before – OpenType in current version 1.9.1. User expectations / satisfaction may be met by drawing design that meets current design, technical and language knowledge. The usage satisfaction relies heavily on intermediate software that renders the font on a layout. This can be anything from simple text processor (TextEdit, Notepad, Word, OpenOffice, ...) to complex layout software (Adobe InDesign, Affinity Designer, Figma, Quark Xpress, Scribus, ...). The font is a software that is plugged in the layout engine and the outlines or features are read from the file, interpreted and presented to the user. It is true that many frustrations coming toward type or usage of type goes on the account of these layout software, because the user experience or interface is not evolving according to standards that are met in other applications we use on daily basis. I do not want to discuss these issues here, so I will focus on design and technical font criteria which can be met during the design or production phase of the font or the font family.

5.2. CRITERIA TO FONT QUALITY

The font may be considered 'quality font' if:

- The design is consistent:

"A mark of a well-crafted typeface is its contouring: How straight are the straight lines, how smooth is the flow of the curves? Secondly, similar characters should appear the same size and positioned uniformly in relation to the

19 ISO 8402, 14:00-17:00. (n.d.). ISO 8402:1994. ISO. https://www.iso.org/standard/20115.html

font lines. Legibility is impaired when characters «dance». Furthermore, the major descriptive elements in the typeface – such as the straight down-strokes, bows and serifs – must present an overall harmonious picture."[20]

- ▷ Uniform stroke weights (unless intentional contrast is part of the design).
- ▷ Harmony across glyph shapes (like curves and angles feeling cohesive, stroke endings, serifs being alike).
- ▷ Diacritics are consistent, shapes are corresponding to local language conventions and are positioned correctly.
- ▷ Outlines are meeting height alignment zones that are defined for lowercase, uppercase, ascender, descender and all other kinds of glyphs (may or may not be figures, smallcaps, sub/supscript) in the font (unless intentional difference in height is part of the design)
- ▷ Outlines are kink-free. Kink is unintentional break in smoothness of two bézier curves connection (unless intentional curve roughness is part of the design).

► The font is legible/readable (not considering screen rendering):

"The primary purpose of a typeface is to transfer information. To attain this goal, it must be legible, i.e. the recipient of the information should be able to decode it with as little effort as possible and reassess the symbols to objects and processes they represent."[21]

- ▷ Well-proportioned letter shapes
- ▷ Use of negative space (in and out) is even
- ▷ Forms are not only recognizable, but also effortlessly read without breaking the flow.

► The font is well spaced and kerned through the character set and properly vertically aligned

"A well-crafted typeface demands good spacing. A state-of-the-art typeface features both traditional spacing (based on total character width) as well as kerning tables, providing correction values for individual character pairs. Where possible, there should exist «kissing» tables for extreme kerning. Good spacing creates text which flows smoothly and rhythmically, enhancing overall legibility."[22]

- ▷ Proper side-bearings (the space around each character).
- ▷ Monospaced fonts are having all glyphs same width (or multiples of base width)
- ▷ Comprehensive kerning classes and pairs. Kerning pair values are consistent and effective in maintaining even distribution of whitespace.
- ▷ Vertical alignment (setting of vertical metrics) of a font file makes the text

20 Karow, P. (2012). Font Technology. Springer Science & Business Media.

21 Karow, P. (2012). Font Technology. Springer Science & Business Media.

22 Karow, P. (2012). Font Technology. Springer Science & Business Media.

centered in a line, the design is taking adequate portion of the UPM and the line spacing is appropriate to the position of caps, diacritics, ascenders and descenders.

- Promoted character sets are completed and correctly encoded
  - Essential character sets (A–Z, a–z, 0–9, punctuation).
  - All supported languages should be complete (accents, diacritics, special symbols), includes “comb” glyphs to enable composition of accented glyphs without Unicode (needed for some scripts/languages)
  - Proper Unicode mappings (correct codepoints assigned for all glyphs)
  - All glyphs are accessible via Unicode or OpeType feature
- The font meets current rendering standards
  - If intended to be used on screen the font includes instructions for rasterizers to display the font crisply ensuring smooth consistent curves and non-jagged lines on both high-res and low-res displays.
- Font family is functional
  - Structure of the font family is clear and the style naming is according to standards.
  - Design axes of the font family are defined and clearly translated to names commonly understood by designers .
  - Font family is installable and all styles are visible to operating system or layout software.
  - If style-linking (having Regular-Bold or Upright-Italic styles) is applicable it is well set up and functional in layout software.

It’s important to note that this list may evolve over time due to technological advancements and changing user expectations. What we are expecting from a typeface today may not be considered important or possible some time ago. There are prevalent qualities like the consistency of design or proper usage of negative space. So are here qualities that evolves like character encodings or screen renderings. What we consider low-resolution display may be in few years viewed like ‘prehistoric’ due to some big technology leap.

5.3. ERRORS, WARNINGS AND INFORMATION MESSAGES

This application is going to run tests on many mentioned qualities or standard compliance and the test results need to be presented to user. The structure of that response must give the user clear message if there is a required following action or not. For this purpose, I will use commonly known principle of errors, warnings and informational messages.

- Error refers to a deviation or non-conformance of the software from specified requirements or guidelines. It also can cause the user to not be able to use or install the software at all. These messages require user response.

- Warning is an exception condition that might not be an error but could cause problems if not addressed. These messages may require user to response.

- Information messages provide feedback about an event in the software that is noteworthy but doesn't require user to take any action.

5.4. CHECK LIST

ROUTINE	DESCRIPTION	RESULT	STATE
META	Name table – if Mac table entries are present than Mac entries = Win entries	ERROR	NOT DONE
META	Name table – entry length (copyright<500, description<200, ID1, ID2, ID4, ID16, ID17 < 64, ID6 < 29, ID18 < 32)	ERROR	DONE
META	Name table – entry length (NID4<31)	WARNING	DONE
META	Name table – NID6 must contain no white spaces	ERROR	DONE
META	Name table – NID4 = NID1 + NID 2 (no Regular)	ERROR	DONE
META	Name table – NID2 can only be one of these four: Regular, Italic, Bold, Bold Italic	ERROR	DONE
META	Name table – NID 4 should start with family name	ERROR	DONE
META	Name table – Make sure family (NID1, NID16) name does not begin with a digit.	ERROR	NOT DONE
META	Style linking – macStyle, NID2 and fsSelection values are consistent	ERROR	DONE
META	italic angle – macStyle, NID2, fsSelection and italicAngle values are consistent	WARNING	NOT DONE
META	Name table – NID1, NID16 must not contain 'Italic'	ERROR	DONE
META	Name table – One family name per family	ERROR	NOT DONE
META	Name table – match the names the proper weight classes	WARNING	NOT DONE
META	Name table – match the names the proper width class	WARNING	NOT DONE
META	OS/2 table – Use Typo Metrics bit is set in fsSelection	WARNING	DONE
META	Version string is the same across meta	ERROR	NOT DONE
META	usWinAscent and usWinDescent matches Ymax and Ymin	ERROR	DONE
META	XavgCharWidth has correct value	ERROR	NOT DONE

META	Check if monospaced font has all glyphs width = avgcharwidth; post. isFixedPitch != 0; kerning is empty	ERROR	NOT DONE
META	font should have these tables: cmap, head, *hhea*, *hmtx*, *maxp*, *name*, OS/2, post	ERROR	NOT DONE
META	*head* table – fontrevision	WARNING	NOT DONE
META	*head* table – UPM <1-16k>	ERROR	NOT DONE
META	*head* table – BoundingBox (Xmin, Xmax, Ymin, Ymax)	ERROR	NOT DONE
META	*hhea* table – advanceWidthMax	ERROR	NOT DONE
META	*hhea* table – minLeftSideBearing	ERROR	NOT DONE
META	*hhea* table – minRightSideBearing	ERROR	NOT DONE
META	*hhea* table – caretSlopeRun	WARNING	NOT DONE
META	*hhea* table – caretSlopeRise	WARNING	NOT DONE
META	*hhea* table – xMaxExtent	ERROR	NOT DONE
META	*hhea* table – numberOfHMetrics	ERROR	NOT DONE
META	Maxp (numGlyphs) = no Glyphs = htmx entries	ERROR	NOT DONE
META	usWeightClass is <1-1000>	ERROR	NOT DONE
META	usFirstCharIndex, usLastCharIndex	ERROR	NOT DONE
META	Code Pages ranges	WARNING	NOT DONE
META	sxHeight – check with x v z y	WARNING	NOT DONE
META	capHeight – check with H	WARNING	NOT DONE
META	GDEF Check mark characters are in GDEF mark glyph class	ERROR	NOT DONE
META	Check GDEF mark glyph class doesn't have characters that are not marks.	ERROR	NOT DONE
META	Check glyphs in mark glyph class are non-spacing.	ERROR	NOT DONE
META	MaxAdvanceWidth is aligned with HMTX table values	ERROR	NOT DONE
CHARSET	charset – encoding list	INFORMATION	DONE
CHARSET	charset – language list	INFORMATION	DONE
CHARSET	charset – encoding list*	ERROR	NOT DONE
CHARSET	glyph order – first four glyphs	ERROR	NOT DONE
CHARSET	custom glyph order*	ERROR	NOT DONE
CHARSET	check if duplicates are the same (Delta – increment, mu – micro, Pi – product)	WARNING	NOT DONE
CHARSET	check if space, CR and nonbreaking space is the same	ERROR	DONE
CHARSET	check if hyphen, nonbreakinghy-phen are the same	ERROR	DONE
CHARSET	Glyphs that are either accessibly via Unicode or feature	WARNING	NOT DONE

CHARSET	Stylistic sets have friendly names	WARNING	NOT DONE
CHARSET	if empty glyphs have no outlines	ERROR	NOT DONE
CHARSET	If glyphs are present out of the private Unicode area	ERROR	DONE
CHARSET	If comb accents exist, then legacy must too	ERROR	NOT DONE
CHARSET	if uppercase exist, check if lowercase exist too and vice versa	WARNING	NOT DONE
CHARSET	all tabular glyphs have same width	ERROR	NOT DONE
GLYPH	leftmost point is in extreme	WARNING	DONE
GLYPH	close to straight line	WARNING	DONE
GLYPH	kinks	WARNING	DONE
GLYPH	overshoot consistency	WARNING	NOT DONE
GLYPH	close to zone	WARNING	NOT DONE
GLYPH	Base glyph component is first	WARNING	NOT DONE
GLYPH	rotated and scaled components	WARNING	DONE
METRICS	Show metric values of single font file	INFORMATION	DONE
METRICS	test metrics of glyphs having the same design	WARNING	DONE
METRICS	Show vertical metrics values and in design	INFORMATION	DONE
METRICS	vertical metrics should be the same *hhea* = typo	WARNING	DONE
METRICS	useTypoMetrics flag is set	WARNING	DONE
KERNING	Show kerning pairs of single font file	INFORMATION	DONE
KERNING	kerning – exists*	ERROR	DONE
KERNING	kerning – positive/negative*	ERROR	DONE DONE
KERNING	kerning – same value*	ERROR	
OT FEATURES	Make all features available to test	INFORMATION	DONE
OT FEATURES	Test if in a stylistic set feature there is not only base glyph, but all accented glyphs too	WARNING	NOT DONE
FAMILY TESTS			
META	weight class – one WWS in family (NID1)	ERROR	DONE
META	style linking – R, I, B, BI only once in style-linking family (NID1)	ERROR	DONE
META	NID 6 is unique in family (whole font family)	ERROR	NOT DONE
META	Underline thickness and position are the same	WARNING	NOT DONE

META	Strikeout position and size are the same	WARNING	NOT DONE
META	Monospaced char width is the same in whole family	ERROR	NOT DONE
META	Same values for Copyright, Version, Trademark, ...	ERROR	NOT DONE
CHARSET	Same charset for the whole family	WARNING	DONE
GLYPH	all styles has the same amount of outlines and components	WARNING	DONE
METRICS	Same Vertical metrics for the whole family	ERROR	DONE
METRICS	Compare metrics of font files in the family	WARNING	DONE
KERNING	Read kerning pairs of the whole font family	INFO	DONE
KERNING	Filter 0 in kerning	INFO	NOT DONE
OT FEATURES	Same features for whole family	WARNING	DONE

Entries marked with * are for custom check only.

6. IMPLEMENTATION

In this section I would like to present the current implementation of the whole application. The application is running on virtual Unix server, and this comes with some more settings than usual web presentation. For the purpose of this work I'm going to describe this part in a simplified manner as the goal of this text is not to precisely give step by step guide how it is done or to document every line of the code.

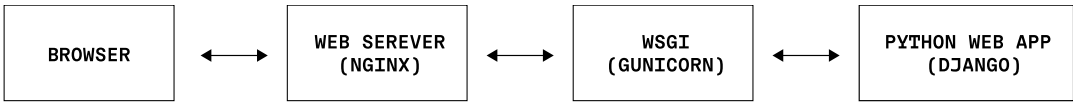
6.1. LOCAL DEVELOPMENT

To begin with such an application, one needs to install proper development environment which is offline and unconnected to the production server. The development server is a server where every application update is written and tested. It must be independent to the production server so nothing can go wrong with what is deployed to the public. There is possibility to install Django on all 3 major operating system – Windows, MacOS, Linux (Unix). For local development on computer, I went with my own computer running native operating system MacOS. There are few recommended steps:

- Install HomeBrew, Python3, Virtual Environment
- Create new virtual environment just for Django
- Install Django
- Install fonttools, xmltodict
- Create new Django project
- Run python manage.py runserver
- Work on 127.0.0.1:8000

6.2. PRODUCTION SERVER

Setting up virtual server is more complicated because there is a configuration of other communication layers needed to be able to get user requests from outside all the way to the python application.



simplified diagram of a structure of a web server serving Django application

The simplified procedure of setting up such a server:

- Update server software
- Install HomeBrew, Python3, Virtual Environment
- Create new virtual environment just for Django
- Install Django
- Install fonttools, xmltodict
- Create new Django project

- Setup Django settings (ALLOWED_HOSTS, DEBUG, SECRET_KEY)
- Manage static and media files
- Database configuration
- Run python manage.py runserver and test it
- Install and setup Gunicorn
- Setup Nginx
- Setup HTTPS and get SSL Certificate
- Firewall setup

For this project I've obtained "fontanalyzer.app" domain and Ubuntu virtual server v22.04. In the time of this writing the server is up and running the latest code.

6.3. READING AND TESTING FONTS

The logic of the whole analysis was described in section 3.5 and here I would like to focus on techniques used for each testing. As mentioned before not all the tests are done. I've focused on completing main tests for each routine (including family) and to showcase maximum variability to the users.

Reading data from the font file is important part of the process where the data are structured to internal dictionary including ID of every data to be tested. Here's a list of function that takes care of that process:

```
readNameTable(inputFont, id)
readCommonTable(inputFont, tableName, id)
readCmapTable(inputFont)
readCmapTableInverse(inputFont)
readGlyphSet(parsedFont, inputFont, id)
readGlyphs(parsedFont, inputFont, id)
readMetrics(parsedFont, inputFont, id)
readKerningPairs(inputFont, id)
readFeatures(inputFont, id)
```

These are run for every uploaded font. Further testing functions for the analysis are requested from the database and run one after each other. Here's a list with brief explanation what every function does and how it fits in the checking list:

```
checkNameTableLengths (font, params)
checkNameTableContains (font, params)
checkNameTableEquals (font, params)
```

These three take care about most of the meta testing of the *name* table. It's either simple comparison of a length of a field to specified length, string comparison if a field contains specified string (or does not contain) or another string comparison if a filed is exactly matching specified string in the parameter. In this function the results are distinguished by the view level.

```
checkStyleLinking (font, params)
```

This function is taking care of testing style linking while looking into *name* (NID1, NID2, NID16, NID17), *head* (macStyle) and OS/2(fsSelection) tables. The setting of the bits and the naming must be aligned. In this function the results are distinguished by the view level.

checkVerticalMetrics (font, params)

This function checks if vertical metrics are aligned in the font and if there is not a node that would exceed the win values. Also the useTypoMetrics bit in fsSelection should be set. This information is shown only to view level 2 and 3.

checkCharsetAgainstList(font, params)

This function is checking the character set of a font with predefined character set. The predefined character set is stored in XML on the server disc. This function takes care of all the charset and language tests. The result is information only and it is accessible to all levels of view.

checkPrivateUnicodeRange(font, params)

This function tests if there is a glyph within private Unicode area (E000-F000). Raise an error. This information is shown only to view level 2 and 3.

checkUnevenLines(font, params)

This function checks all the curves for unevenness. If the font is inclined (italic, slanted) it tries to compute angle of the line and compares it to the italic angle defined in post table. The angle which is considered as close to straight is by default from 0.8 degree to 5 degrees. If the line is very short the upper limit is double or even quadruple of the default. (It's very easy to get angle difference with thin styles on a short line, because of the low granularity of the grid). This function gives all the view levels the same information.

checkKinks(font, params)

This function checks for unintentional breaks in curve smoothness. The kink is considered to be present when the angle of two consequent handles is differing more than 0.5 degrees to 15 degrees. If one of the handles is 7 times longer than the other, the kink is very likely to be present (cannot be easily avoided because of the low granularity of the grid) and therefore it is ignored. This function gives all the view levels the same information.

checkLeftmostExtreme(font, params)

This function is checking if the leftmost node is oncurve (node, not handle). This function gives all the view levels the same information.

checkComponentRotationScale(font, params)

This function is checking if one or more components are not rotated or scaled. It may not cause a trouble during rendering, but if any hinting is applied to the base glyph, the transformation on it is not applied. For example, delta instruction of moving a point up will be the same in mirrored glyph and thus also moving up and causing rendering problem. This is all just prevention of a future problems, and it is flagged as warning. This function gives all the view levels the same information.

checkMetrics (font, params)

This function tests metrics of a glyph against predefined sign, value or reference. The predefined sign can be either negative or positive, value can be any

integer value and reference can be any name or Unicode. This information is shown only to view level 2 and 3.

checkKerningPairsExist(font, params)
checkKerningPairsPolarity(font, params)

These two functions are checking kerning pairs for existence or polarity (negative/positive). Kerning pairs are defined as pair of Unicodes. These functions can be run only in custom view level 2 or 3.

checkOTFeatures(font, params)

This function check whether the specified OpenType feature is present. The name of the feature is defined as 4 letter string. This function can be run only in custom view level 2 or 3.

checkFamilyWeightClasses(fonts, params)

This function is checking if there are two fonts with the same WWS and at the same time not having fsSelection flag Regular set. This translates to raising an error if there would be two Italic, Bold or Bold Italic fonts in style linking family. This function gives all the view levels the same information.

checkFamilyLanguages(fonts, params)

This function checks if there is the same language support in the family. This function gives all the view levels the same information.

checkGlyphsConsistency(fonts, params)

This function checks if name corresponding glyphs have the same number of paths and contours. This function gives all the view levels the same information.

checkFamilyVerticalMetrics(fonts, params)

This function checks if vertical metrics are the same through the whole font family. This information is shown only to view level 2 and 3.

checkFamilyHorizontalMetrics(fonts, params)

This function computes average delta of metrics from one to another style and if there is a bigger deviation (5 times) from that the metrics are flagged suspicious. This information is shown only to view level 2 and 3.

checkFamilyOTFeatures(fonts, params)

This function checks if there are the same OT features in the family. This function gives all the view levels the same information.

The results are than saved back to database.

Right after this phase the presentation data are prepared, errors and warning counts are computed, and everything is sent to the template corresponding to the view level.

6.4. LOGGING

In order to inform user about all the tests that were run on uploaded fonts logging was recently added to the analysis. The log is a simple text filed where the application can note what is done during the process of testing. This text output is than connected to the left upper information box called summary. Clicking on any of the first three values takes the user to the page where the log is displayed.

The log also allows me as a developer to note the precise time of any action taken during the testing and thus improve performance. This will be briefly discussed in next chapter

6.5. SPEED

The speed of server-side processes must be tracked, and each function must be evaluated to find most time demanding computations. With rising number of tests, the total time will be rising, and it may degrade the user experience of the app.

During the development I took some measures to make the process more faster by moving call for one function in many tests to initial reading phase. The results of the function were saved to dictionary and then simply read by in these functions. At this point it seems that the most time-demanding function is reading *cmap* table which takes nearly 1 second.

Also to make user experience a bit better (less unexpected) I've added waiting message after the upload and before the presentation starts to render. It does not improve any time measures, just prepares the user to wait maybe a bit longer.

Current version takes around 30 seconds to test 6 fonts family.

7. CONCLUSION

This thesis examines very specific field of font engineering and gives better perspective on the challenges and tasks the font engineer must undergo to produce quality font. Through broad research of literature, public questionnaire answered by professionals and 10 years of personal experience the thesis could state what the current definition of quality font is.

Based on the definition and modern software possibilities whole new application on font quality assurance was designed, programmed and deployed on real functional server. The application goals were met.

Font analyzer is running on any up to date desktop machine running any operating system. It is possible to insert opentype fonts exported from any current font development software. The application is based on Django – existing framework, which made the development much easier and straightforward. The application is easy to expand with tests, tests definitions or new functionality gained from the underlying font tools library. The application is free too use and it will stay free. The application is able to test multiple fonts in one run and also to provide font family testing, which test all the fonts as a family. The application was designed with common usability principles taken into account and tested by real users. The conclusions from testing were taken into account and the design was improved. During the development all kinds of existing fonts were tested as an input – students fonts, type designers fonts and even font of large font production houses. In every step of the development of this application the were other font engineers and type designers involved. In the very beginning the conducted questionnaire gave my important insight how other work, but also the reflection of other user behavior during design testing was important to me. Few engineers also gave me ideas to implement new tests that otherwise would stay only in their wishes.

The development of the application is not done. This thesis is a valuable first step to build software that could serve as a useful basepoint to many beginners, type designers or font engineers. This thesis also could serve as a starting point to write better documentation to quality assurance process in the font industry and to revisit and discuss what we are actually doing and how to improve it.

The future for this application lays mainly in programming the rest of the tests from the check list and to broaden the testing to new emergning variable and color fonts. Improve speed and effectivity of computations and build comprehensive documentation to these tests. In longer run fonts from around the world could be brought in and the testing and usability can be adjusted to different scripts and writing systems. In broader view this application can serve not only individuals, but also to schools, workshops and type foundries.

8. APPENDIX

8.1. ABBREVIATIONS

- ANSI = American National Standards Institute
- CGA = Color Graphics Adapter
- FE = Font Engineer
- HOI = Higher Order Interpolation
- HTML = Hyper Text Markup Language
- HTTP = Hyper Text Transfer Protocol
- HTTPS = use of HTTP over SSL/TLS
- IP = Internet Protocol
- ISO = International Standard Organization
- NDA = Non-Disclosure Agreement
- NID = Name Table ID
- OS = Operating System
- OT Feature = OpenType Feature
- OTF = OpenType Font
- PPEM = Pixel Per EM
- QA = Quality Assurance
- SSL = Secure Sockets Layer
- SVG = Scalable Vector Graphics
- SW = SoftWare
- TD = Type Designer
- TTF = TrueType Font
- UPM = Units Per EM
- URL = Uniform Resource Locator
- UX = User Experience
- WWS = Weight, Width, Style
- XML = eXtensible Markup Language

8.2. BIBLIOGRAPHY

- Karow, P. (2012). Font Technology: Methods and Tools (Softcover reprint of the original 1st ed. 1994 ed.). Springer.

- Brinck, T., Gergle, D., & Wood, S. D. (2002). Designing Web sites that work : usability for the Web. Morgan Kaufmann Publishers.

- Website Usability: The Ultimate Guide for 2024. (n.d.). Survicate.com. https://survicate.com/blog/website-usability/

- Interaction Design Foundation. (2016, September 25). What is Wireframing? The Interaction Design Foundation; Interaction Design Foundation. https://www.interaction-design.org/literature/topics/wireframe

- Frontend and backend. (2021, December 20). Wikipedia. https://en.wikipedia.org/wiki/Frontend_and_backend

- Font Names Table – TrueType Reference Manual – Apple Developer. (2020). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6name.html

- Horizontal Header Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6*hhea*.html

- Font Header Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6head.html

- OS/2 Compatibility Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6OS2.html

- Glyph Name and PostScript Font Table – TrueType Reference Manual – Apple Developer. (2025). Apple.com. https://developer.apple.com/fonts/TrueType-Reference-Manual/RM06/Chap6post.html

- ISO 8402, 14:00-17:00. (n.d.). ISO 8402:1994. ISO. https://www.iso.org/standard/20115.html