Web application for ordering and automatic printing of custom stickers

Peter Basár
Czech technical university in Prague
Faculty of Electrical Engineering
Department of Control Engineering

May 20, 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Basár**	Jméno: **Peter**	Osobní číslo: **492320**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra řídicí techniky**		
Studijní program:	**Kybernetika a robotika**		

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Webová aplikace pro objednávání a automatický tisk nálepek na míru

Název bakalářské práce anglicky:

Web application for ordering and automatic printing of custom stickers

Pokyny pro vypracování:

Cílem bakalářské práce je aplikace, která umožní uživatelům objednávat vlastní motivy nálepek v malém množství, které budou zpracované pro automatizovanou výrobu. Východiskem bakalářské práce je rešerše stávajících řešení a komunikace se společnostmi o problému tisku v malém počtu. Výsledkem bakalářské práce také bude jak zpracování vstupu pro tisk, tak i vhodného vstupu pro automatizované vyřezávání z daného tisku.
Implementace webové aplikace, jak serverové tak uživatelské strany, bude umožňovat uživatelům nahrát si vlastní obrázky, vybrat si z existujících a výrobci tyto různé uživatelské objednávky předá ve vhodném formátu.
Navrhněte způsob testování aplikace, proveďte test aplikace a výsledky testu vyhodnoťte.

Seznam doporučené literatury:

[1] Feldroy D., Feldroy A., Two scoops of django 3.x
[2] Oficiální stránka knihovny React reactjs.org
[3] Oficiální dokumentace Django docs.djangoproject.com

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Ivan Jelínek, CSc. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce:
do konce letního semestru 2022/2023

doc. Ing. Ivan Jelínek, CSc.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank doc. Ing. Ivan Jelínek, CSc. for his guidance and giving me this oportunity, Department of control engineering for enabling me to work on this thesis and my friends and family for supporting me during my studies.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstract

Automation of ordering and production systems is one of the ways that can be used to make products more widely accessible among the customers by lowering the production cost and prices.

This work focuses on automation in the area of custom sticker printing. However, this work could be generalized to other industries that deal with custom ordering with products such as glass, wood, engraving, or clothing.

Automation is being implemented using a web application, which takes care of the communication with customers and provides data (images and dimensions) to production processes such as printing and cutting.

Research of existing solutions, both local and foreign companies, is introduced at the beginning of this work. Additionally, this work contains the functionality description and design of both the frontend and backend of the application. Application is also fully implemented, and the source code can be found in the bibliography [20]. The backend utilizes Python (framework Django [1]) and database PostgresSQL [4], and the frontend uses JavaScript language (framework React [3]). Moreover, the application is also deployed on a publicly accessible server and was tested in automated printing. However, a small home printer was used in automated printing, not the industry-grade printer and a cutter used in actual production. The printer printed sticker images and their cutout image masks beneath each other according to the order made by the user.

In conclusion, it can be stated that the goals of the bachelor thesis were met. An application has been created that will allow users to order sticker designs in small quantities for automated production. A survey of existing solutions was carried out (Section 2.3), and the issue of small-scale printing was specified in communication with companies. The core of the bachelor's thesis is also the processing of input for printing and cutting, the editor for uploading designs, and handing over the design to the manufacturer in a suitable format. Part of the bachelor's thesis is the final testing of the application, including the evaluation of results.

The application will be publicly accessible on address `sticker-application-frontend.herokuapp.com` and `sticker-application-backend.herokuapp.com` for at least one year. For possible changes or problems, refer to the source code[20]

Abstrakt

Automatizace objednávacích a výrobních procesů je jeden ze způsobu jakým můžeme rozšířit dostupnost produktů mezi zákazníky pomocí snižování nákladů a cen.

Táto práce se zabývá automatizací v problematice tištění nálepek na zakázku, ale je jí možné zobecnit i na další odvětví, které se této problematice objednávaní na zakázku blíží, jako například objednávaní skla, dřeva, gravírování nebo oblečení.

Automatizace je docílená využitím webové aplikace, která se stará o komunikaci se zákazníky jako i o vhodné předávání dat (obrázky a rozměry) výrobním procesům jako jsou tištění a řezání.

Součástí práce je průzkum existujících řešení a společností jak lokálních, tak i zahraničních. Dále také obsahuje popis funkcionality, návrh uživatelské i serverové strany aplikace. Kromě návrhu je taky plně implementovaná a zdrojový kód se nachází ve výčtu zdrojů práce[20]. Serverová část využívá jazyku Python (framework Django [1]) a databázi PostgresSQL [4], uživatelská strana využívá jazyku JavaScript (framework React [3]). Na závěr je aplikace taky spuštěná na webu a testovaná v automatickém provozu, kde po vytvoření objednávky dochází k automatickému tištění. Pro tištění nejsou využitý skutečný tiskárny používány na nálepky ani řezací stroje ale využívá se pouze domácí tiskárny, která vytiskne vizuální návrh nálepky a taky masku vyřezání v požadované velikosti.

Ačkoli je aplikace na webu tak je veřejně dostupná a bude k dispozici nejméně rok po odevzdání této práce. Uživatelská a serverová část aplikace je dostupná na adrese `sticker-application-frontend.herokuapp.com` a `sticker-application-backend.herokuapp.com`. Při problémech s připojením doporučuji nahlédnout do zdrojového kódu aplikace[20], který bude potencionálně aktualizován.

Na závěr může být řečeno, že cíle bakalářské práce byly splněny. Aplikace, která umožní uživatelům objednávat nálepky v malém množství s automatickým tištěním. Byl proveden průzkum existujících řešení (Section 2.3), přičemž bylo dotazováno na problematiku tisku malého počtu nálepek na objednávku. Mezi jádro bakalářské práce patří zpracování vstupu na tištění a ořezávání, editor pro nahrávání obrázků, a předávání objednávky výrobci ve vhodném formátu. Část bakalářské práce tvoří také finální testování aplikace a zhodnocení výsledků.

Klíčová slova: bakalařská práce, webová aplikace, autom-

Keywords: bachelor's thesis, web application, automation, automated printing, custom orders, stickers, backend, frontend, Python, Django framework, Django REST framework, API, React framework, JavaScript, database, PostgresSQL, JWT, filtration, editor

atizace, automatický tisk, objednávání na míru, nálepky, serverová strana, uživatelská strana, Python, Django framwork, Django REST framework, API, React framework, JavaScript, databáze, PostgresSQL, JWT, filtrace, editor

Překlad názvu: Webová aplikace pro objednávaní a automatický tisk nálepek na zakázku

Contents

1 Chapter: Introduction

1.1 motivation

Software used for automating communication with customers can be a powerful tool for making it easier for customers to solve their requests quickly and for the company to provide a particular service promptly.

The research, design, implementation, and testing of such an application is the purpose of this thesis.

Users can upload their design files or order and slightly modify existing ones from the gallery with this application. Customization is enhanced by providing users with a web editor for uploading and editing their designs.

Additionally, this application's approach can be used to automate other services as well. Services that require visual customization with user-provided designs can utilize the entirety of this application. Services that are not highly customizable can mainly use the technique of processing customer orders.

Examples of such services can include various clothing customization applications (shoes, t-shirts, headwear), sporting equipment (snowboard, skateboard, skiing equipment), editing custom furniture dimensions and or shapes, and many more.

1.2 results

The results of this thesis are a walkthrough of the entire design process of the frontend and the backend of the application, implementation of both frontend and backend as well as testing the functionality of the application.

The design is supported by the research of existing solutions and discussions about those solutions.

2 Chapter: Existing solutions

Research has been done in December 2021 and existing solutions may have changed since that period.

2.1 good examples

2.1.1 Redbubble [8]

great filtering system, lots of options, amazing presentation

Among the best companies that sell stickers with vast designs is Redbubble.

With headquarters in Australia, this company offers a vast ecosystem where people can, among other things, order stickers in 3 different sizes from the gallery with hundreds of thousands of designs.

A Filtering

For customers not to get lost in this extensive gallery, Redbubble offers an exemplary filtering system, and from my observation, it works as follows.

After the visitor reaches the product listing page, he will be offered different keywords which are used to describe stickers Figure 1.

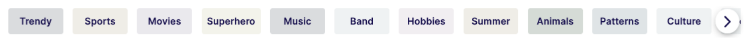


Figure 1: Initial filtering options

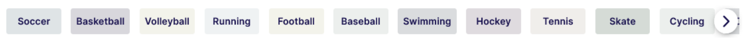


Figure 2: Filtering options after selecting keyword "Sports"

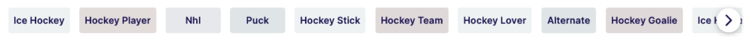
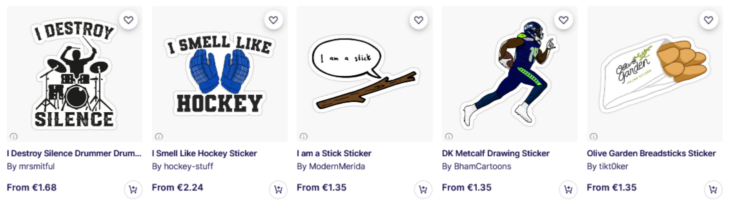


Figure 3: Filtering options after selecting "Hockey"

We can observe that to the right side of keywords in Figures 1, 2 and 3 there is an arrow with which a user can access many more keywords. Categories are often similar, but they also seem a bit random, meaning that they are probably not part of a hand-crafted structure but rather it is a listing of the most common keywords which are related to the one that the user is currently browsing and that after selecting the new keyword, the previous selections are forgotten. A bit of randomness and unrelatedness can be seen in Figure 4 which is a result of selecting keywords in order "Sports", "Hockey", "Sticks" and that about half of the filtered stickers aren't associated with the first two categories, but only the last one.



I Destroy Silence Drummer Drum...
By mrsmitful
From €1.68

I Smell Like Hockey Sticker
By hockey-stuff
From €2.24

I am a Stick Sticker
By ModernMerida
From €1.35

DK Metcalf Drawing Sticker
By BhamCartoons
From €1.35

Olive Garden Breadsticks Sticker
By tikt0ker
From €1.35

Figure 4: Filtration options after selecting categories in order: "Sports" - "Hockey" - "Sticks"

Additionally, filtering also changes URL suffix by adding "/hocker+sticker" resulting in URL `https://www.redbubble.com/shop/hockey+stickers`. By adding "stickers" in the suffix, the site also filters out many of the products besides stickers it offers.

B Product page

After clicking on the sticker, we are redirected to its page. There, the presentational images are trying to inform the customer about the size of the chosen sticker. There are a total of 4 sizes per sticker (Figure 5) and 3 items on which the sticker is being placed to present its size: water bottle, laptop, and bedside table (Figure 6). Additionally, there are three material options to choose from: matte, glossy, and transparent. Each of these items is also appropriately presented Figure 7.

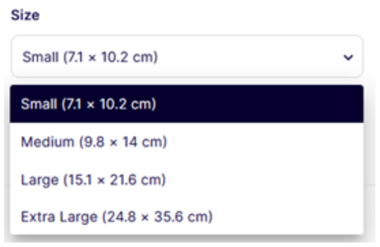


Figure 5: Sticker size options

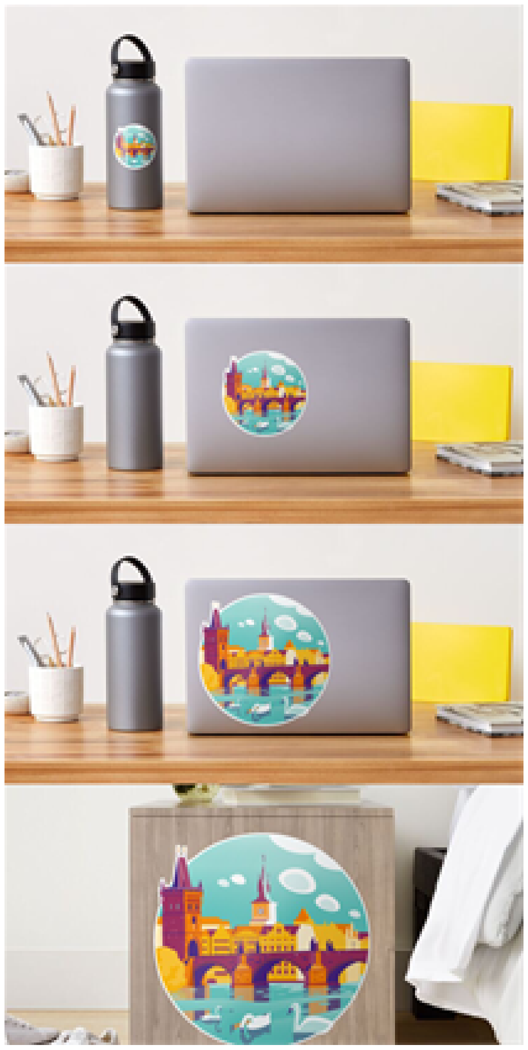


Figure 6: Size presentation on familiar items

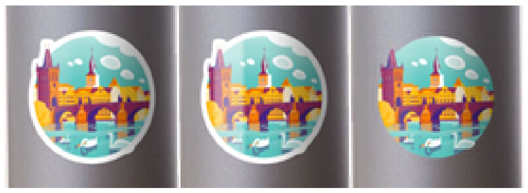


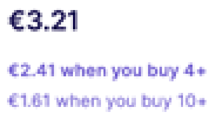
Figure 7: Presentation of chosen materials, from the left side: matte, glossy, and transparent

By having this presentation, visitors are being helped with imagining the size of the sticker they might order. One of the disadvantages, however, can be that a customer is limited to only four size options.

After a quick inspection of the site, it can be observed that the images which are representing the size are unique for each of the chosen sizes and materials. Those images are most likely static images. By having static images for size representation, they are limiting themselves for adding features such as giving users options to pick the size they want, because if they were to give users 100 size options, the space required to store all of these images would be vast and having server-side generated images on request is also not computationally viable.
One way that this could be solved would be to have the frontend pair those images together by simply positioning them on top of each other, or more comfortably using one of the frontend packages to combine them into one image, for example, ”merge-images” Node.js package [7].

Different materials also have their reflective properties (Figure 7), and after closer inspection, for one sticker, the site needs 25 images of surprising size 1.84 MB [1], surprising because I was expecting even bigger space needed.

The site also offers a lower price per sticker (Figure 8), unfortunately only after ordering more of the same type. The price per one small-sized sticker is about 3.21€, which is relatively expensive.
Additional features that the site offers are user reviews, a small description of the chosen material, slidable sticker listing by the same creator, another slidable sticker listing with similar designs, and a list of all tags used by the sticker (Figure 9).

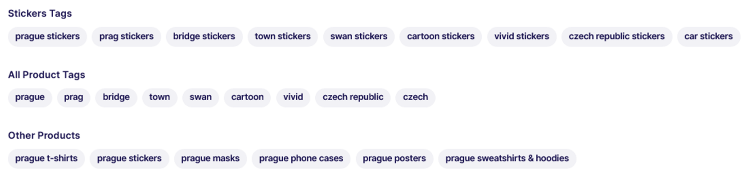

€3.21

€2.41 when you buy 4+

€1.61 when you buy 10+

Figure 8: Information about the lower price after buying more stickers of the same type

[1]Checked by downloading the source images for one material and one size and then multiplying by 12: sizes*mateirals


Stickers Tags

prague stickers · prag stickers · bridge stickers · town stickers · swan stickers · cartoon stickers · vivid stickers · czech republic stickers · car stickers

All Product Tags

prague · prag · bridge · town · swan · cartoon · vivid · czech republic · czech

Other Products

prague t-shirts · prague stickers · prague masks · prague phone cases · prague posters · prague sweatshirts & hoodies

Figure 9: All tags/keywords used by the chosen sticker

2.1.2 Diginate [9]

great filtering system, lots of options, amazing presentation, absence of existing designs

Another good example is the website Diginate, which resides in England and offers only printing of your uploaded designs.

A Materials

Diginate offers vinyl, metallic, and windows or cling sticker types (Figure 10). Each of these options offers different materials.
For vinyl: gloss, matt, clear, textured, and wall.
For metallic: silver, gold, brushed aluminum, brushed gold, sparkly, holographic.
For cling stickers: gloss, clear, gloss self-cling, clear self-cling.
Visitors can choose from 15 different types of stickers and are offered quality presentational images and gradual options to not get lost in types of stickers.

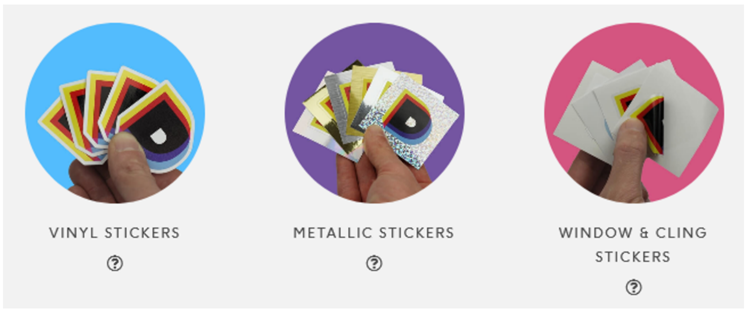


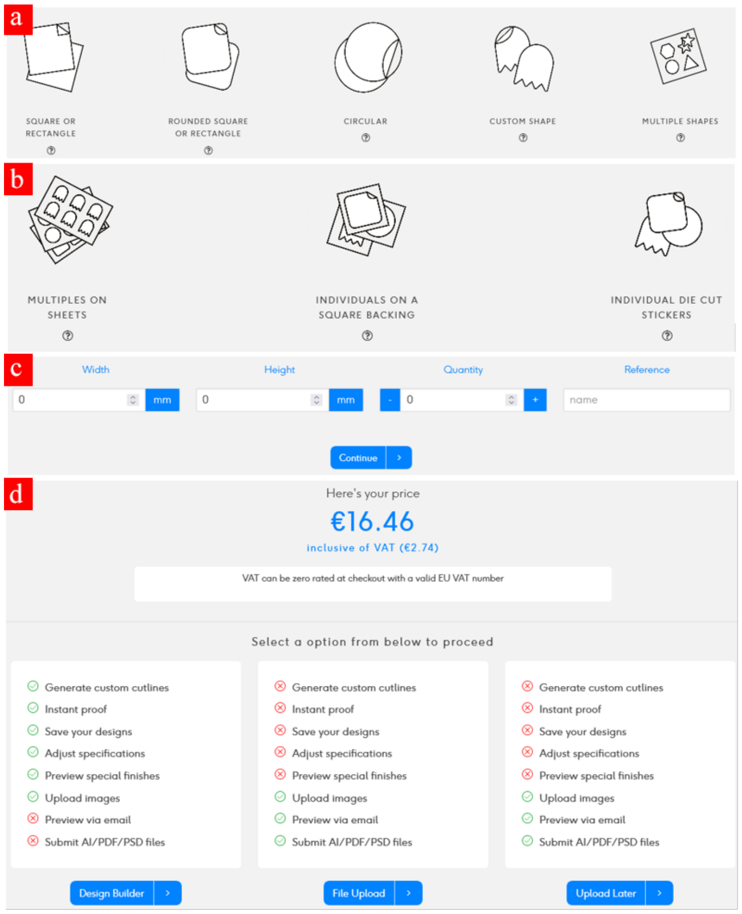
Figure 10: Different sticker types offers from Diginate

B Editor

If users want to upload their stickers, they are recommended to use the existing editor[2] which they can reach within

[2]in option ’d’ in Figure 11 over the ”Design Builder” option, there are the biggest amount of checkmarks

a few clicks: choosing sticker type, choosing material, the shape of the sticker, sticker finalization[3], setting sticker dimensions and then clicking on the "Design builder" (Figure 11).



Width	Height	Quantity	Reference
0 mm	0 mm	- 0 +	name

Here's your price
€16.46
inclusive of VAT (€2.74)
VAT can be zero rated at checkout with a valid EU VAT number

Select a option from below to proceed

✓ Generate custom cutlines	✗ Generate custom cutlines	✗ Generate custom cutlines
✓ Instant proof	✗ Instant proof	✗ Instant proof
✓ Save your designs	✗ Save your designs	✗ Save your designs
✓ Adjust specifications	✗ Adjust specifications	✗ Adjust specifications
✓ Preview special finishes	✗ Preview special finishes	✗ Preview special finishes
✓ Upload images	✓ Upload images	✓ Upload images
✗ Preview via email	✓ Preview via email	✓ Preview via email
✗ Submit AI/PDF/PSD files	✓ Submit AI/PDF/PSD files	✓ Submit AI/PDF/PSD files
Design Builder	File Upload	Upload Later

Figure 11: Nested options a users has to go through in order to get to the editor

After choosing the golden material, the background of the editors is presented as golden. After uploading a transparent sticker, the golden background is visible through the transparent part of the sticker.
The editor has an option to create a cutout border automatically[4]. I am not sure as to what type of algorithm they are using for generating those cutout lines, but after observing how this algorithm behaves, it may be that each of the non-filled spaces in the sticker is used as a starting point for the cutout-line generator and is expanded to the nearest sticker border (Figure 12).
Users can then choose which of these cutout lines they want to use and how far away from the sticker they want to have the line (Figure 13).
Unfortunately, users cannot create these cutout lines individually.
After cutting out the lines and saving the motive[5], users are transferred to the presentational site where they can slightly rotate the sticker with their cursor and change the reflective properties, which is great complementary detail and gives the users the feeling of familiarity as they can somewhat observe the sticker in 3D (Figure 14).

The biggest downside of this editor is, however, that it is very slow. After editing the cutout lines of the sticker, it takes roughly 3 seconds to see any difference and the experience is very frustrating. If this wasn't the case, this editor would be very close to being perfect.

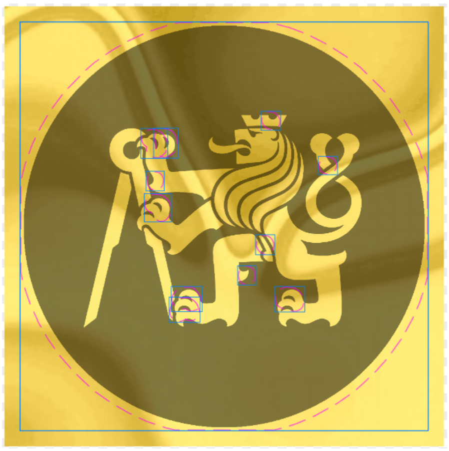


Figure 12: Automatically generated cutout lines offered to the user

[3]whether users wants stickers on a sheet of paper or have them cut
[4]only if the user chose the editor with custom cut-out shape

[5]user has to register

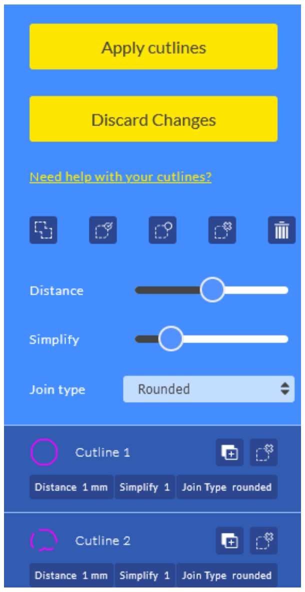


Figure 13: Options user has for editing the generated cutout lines

Figure 14: Presentation of the sticker after applying cutout lines and continuing to the next stage of order fulfillment. Presentation is interactive and slightly rotates with the mouse movement, which also changes its reflectiveness

2.1.3 Stickeryou [10]

speed and simplicity of the editor
With Stickeryou, customers can shop for multiple products, such as magnets, t-shirts, patches, and stickers.

A Editor

Stickeryou doesn't provide ordering of existing motives. However, they are utilizing a very simple form of an editor, which may seem limiting and simplistic, but it is the fastest one and offers one of the best user experiences.

The uploaded image is fitted into one of the chosen cutout motives, or a border is created evenly around a sticker after using the "Image Die-Cut" option, which takes into account the transparency of the image Figure 15. Additional editing options include also resizing the image.

One small problem, however, is that when there is text in the uploaded image. The "Image Die-Cut" option creates unwanted borders, and there is no option for a user to correct this behavior Figure 16. However, a more experienced user can create a white background around the text in his local editor and then upload the image.

Additionally, they put heavy emphasis on printing in a big volume as the price for a small sticker (3cm x 3cm) is roughly 9€.

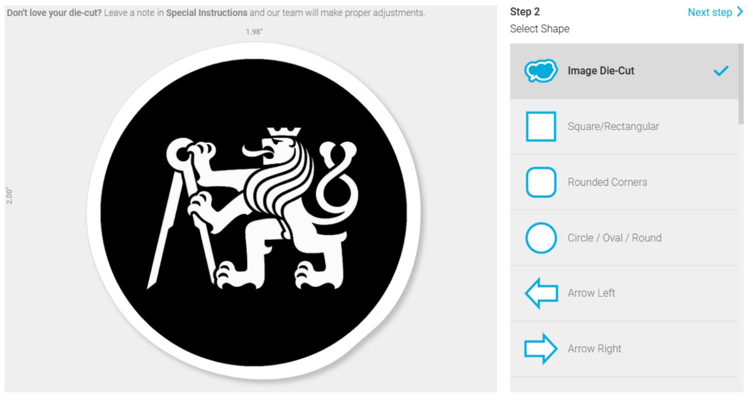


Figure 15: Sticker cut-out shape selection options

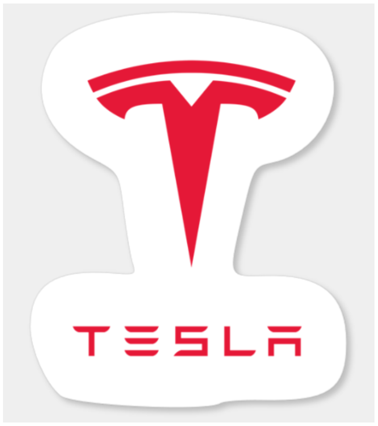


Figure 16: Undesirable cut-out border for transparent stickers with text

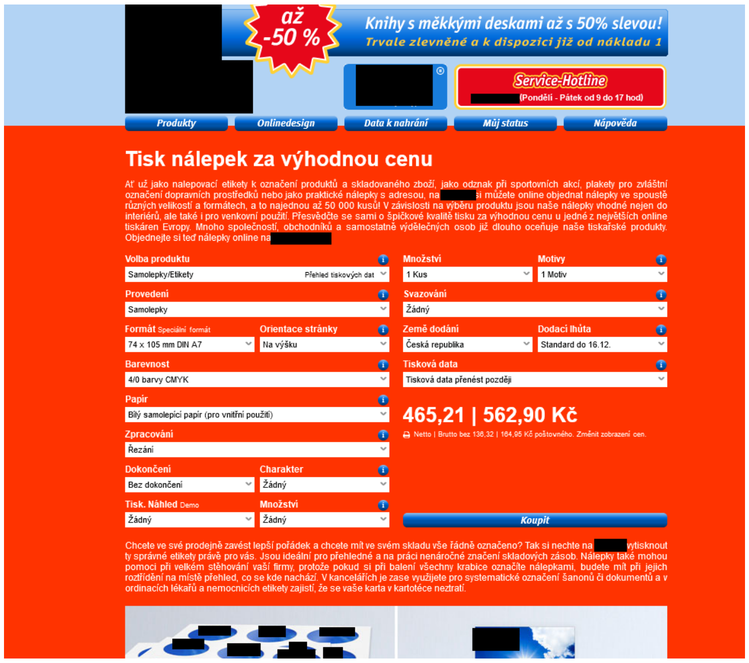
2.2 worse examples

Companies that aren't presented in a positive light are anonymized with an assigned number in Figure 19. At least one company mentioned in this section improved their website during the time this work was being finalized, and specifically naming the company is not essential for this work.

When it comes to local companies, I've found only one among the other 19, which offers users an editor for uploading sticker design, and that is Expressprint.cz. This editor, however, is not fully-featured. After the user uploads the image into the designated rectangular area, he can only rotate and resize the sticker. The type of cutout has to be specified in the text area with the order.

The prices which all the local companies offer are high enough that they would discourage any customer that wants to print a few stickers. Either these prices aren't publicly available, or one needs to make an order of 10, 50, or 100 stickers with the same motive, or the prices go upwards of 12€ per sticker. But as expected, these prices come to reasonable levels but only after a large order has been made. Moreover, vast majority of local companies also utilizes a rather old-fashioned web-design for example Figures 17 and 18, not so enchanting presentational images Figure 18, or a very lenthy way of ordering sticker using the text form[6].

Additional data about more companies can be viewed in the following table Figure 19.


[redacted]

až -50 %

Knihy s měkkými deskami až s 50% slevou!
Trvale zlevněné a k dispozici již od nákladu 1

[redacted]

Service-Hotline
[redacted] (Pondělí - Pátek od 9 do 17 hod)

Produkty | Onlinedesign | Data k nahrání | Můj status | Nápověda

Tisk nálepek za výhodnou cenu

Ať už jako nalepovací etikety k označení produktů a skladovaného zboží, jako odznak při sportovních akcí, plakety pro zvláštní označení dopravních prostředků nebo jako praktické nálepky s adresou, na [redacted] si můžete online objednat nálepky ve spoustě různých velikostí a formátech, a to najednou až 50 000 kusů! V závislosti na výběru produktu jsou naše nálepky vhodné nejen do interiérů, ale také i pro venkovní použití. Přesvědčte se sami o špičkové kvalitě tisku za výhodnou cenu u jedné z největších online tiskáren Evropy. Mnoho společností, obchodníků a samostatně výdělečných osob již dlouho oceňuje naše tiskařské produkty. Objednejte si teď nálepky online na [redacted]

Volba produktu: Samolepky/Etikety — Přehled tiskových dat
Množství: 1 Kus
Motivy: 1 Motiv
Provedení: Samolepky
Svazování: Žádný
Formát Speciální formát: 74 x 105 mm DIN A7
Orientace stránky: Na výšku
Země dodání: Česká republika
Dodací lhůta: Standard do 16.12.
Barevnost: 4/0 barvy CMYK
Tisková data: Tisková data přenést později
Papír: Bílý samolepící papír (pro vnitřní použití)
Zpracování: Řezání
Dokončení: Bez dokončení
Charakter: Žádný
Tisk. Náhled Demo: Žádný
Množství: Žádný

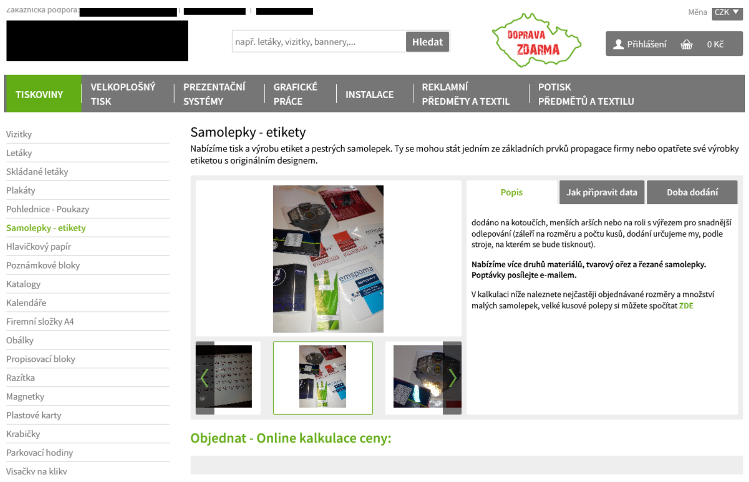
465,21 | 562,90 Kč

Netto | Brutto bez 136,32 | 164,95 Kč poštovného. Změnit zobrazení cen.

Koupit

Chcete ve své prodejně zavést lepší pořádek a chcete mít ve svém skladu vše řádně označeno? Tak si nechte na [redacted] vytisknout ty správné etikety právě pro vás. Jsou ideální pro přehledné a na práci nenáročné značení skladových zásob. Nálepky také mohou pomoci při velkém stěhování vaší firmy, protože pokud si při balení všechny krabice označíte nálepkami, budete mít při jejich roztřídění na místě přehled, co se kde nachází. V kancelářích je zase využijete pro systematické označení šanonů či dokumentů a v ordinacích lékařů a nemocnicích etikety zajistí, že se vaše karta v kartotéce neztratí.

Figure 17: Web-design of Company #22


Zakaznicka podpora: [redacted] | [redacted] | [redacted]

Měna CZK

[redacted]

např. letáky, vizitky, bannery,... Hledat

DOPRAVA ZDARMA

Přihlášení 0 Kč

TISKOVINY | VELKOPLOŠNÝ TISK | PREZENTAČNÍ SYSTÉMY | GRAFICKÉ PRÁCE | INSTALACE | REKLAMNÍ PŘEDMĚTY A TEXTIL | POTISK PŘEDMĚTŮ A TEXTILU

Vizitky
Letáky
Skládané letáky
Plakáty
Pohlednice - Poukazy
Samolepky - etikety
Hlavičkový papír
Poznámkové bloky
Katalogy
Kalendáře
Firemní složky A4
Obálky
Propisovací bloky
Razítka
Magnetky
Plastové karty
Krabičky
Parkovací hodiny
Visačky na kliky

Samolepky - etikety

Nabízíme tisk a výrobu etiket a pestrých samolepek. Ty se mohou stát jedním ze základních prvků propagace firmy nebo opatřete své výrobky etiketou s originálním designem.

Popis | Jak připravit data | Doba dodání

dodáno na kotoučích, menších arších nebo na roli s výřezem pro snadnější odlepování (záleří na rozměru a počtu kusů, dodání určujeme my, podle stroje, na kterém se bude tisknout).

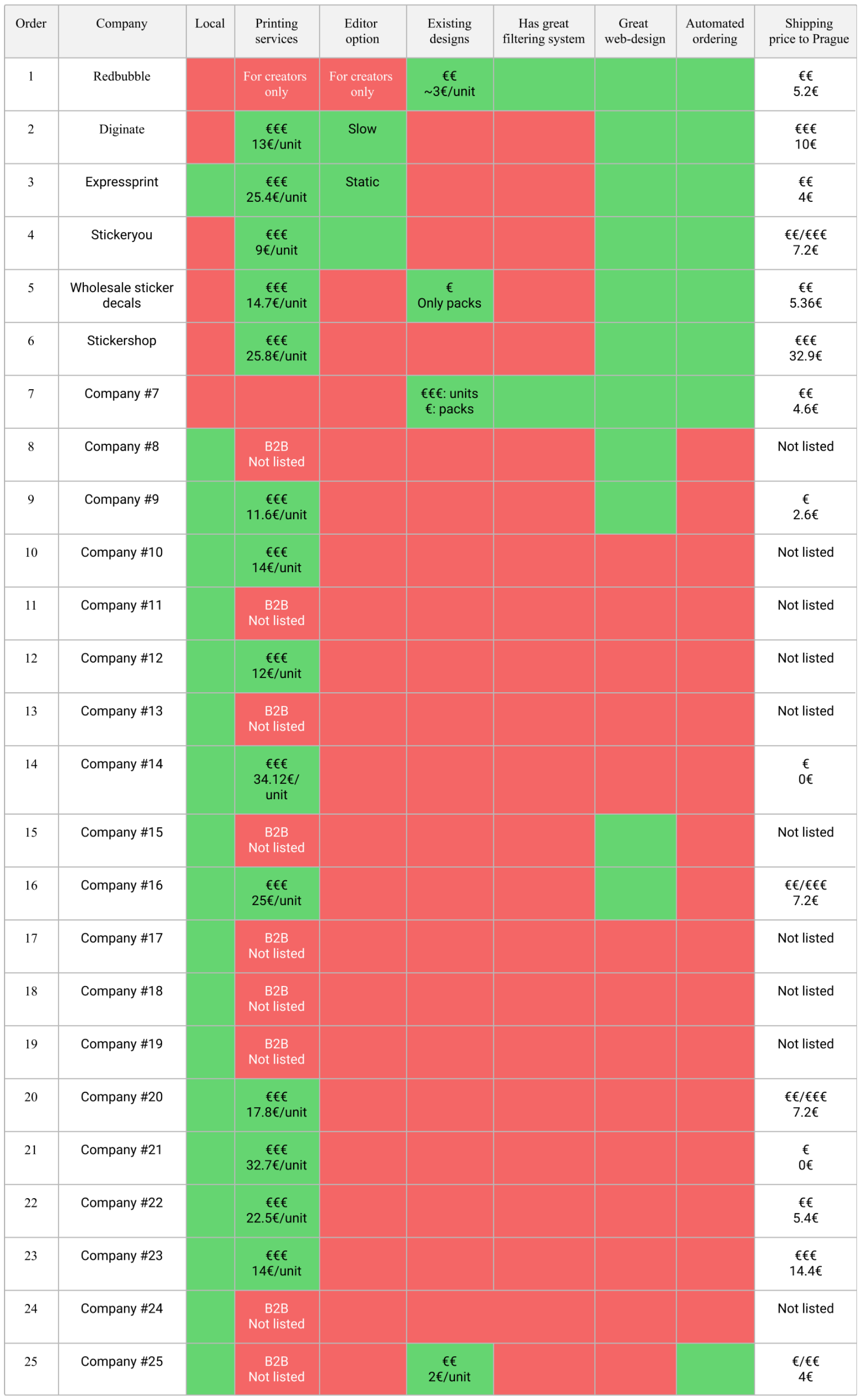
Nabízíme více druhů materiálů, tvarový ořez a řezané samolepky. Poptávky posílejte e-mailem.

V kalkulaci níže naleznete nejčastěji objednávané rozměry a množství malých samolepek, velké kusové polepy si můžete spočítat ZDE

Objednat - Online kalkulace ceny:

Figure 18: Web-design of Company #21

[6]this applies to most of the little local companies



Order	Company	Local	Printing services	Editor option	Existing designs	Has great filtering system	Great web-design	Automated ordering	Shipping price to Prague
1	Redbubble	🟥	🟥 For creators only	🟥 For creators only	🟩 €€ ~3€/unit	🟩	🟩	🟩	€€ 5.2€
2	Diginate	🟥	🟩 €€€ 13€/unit	🟩 Slow	🟥	🟥	🟩	🟩	€€€ 10€
3	Expressprint	🟩	🟩 €€€ 25.4€/unit	🟩 Static	🟥	🟥	🟩	🟩	€€ 4€
4	Stickeryou	🟥	🟩 €€€ 9€/unit	🟩	🟥	🟥	🟩	🟩	€€/€€€ 7.2€
5	Wholesale sticker decals	🟥	🟩 €€€ 14.7€/unit	🟥	🟩 € Only packs	🟥	🟩	🟩	€€ 5.36€
6	Stickershop	🟥	🟩 €€€ 25.8€/unit	🟥	🟥	🟥	🟩	🟩	€€€ 32.9€
7	Company #7	🟥	🟥	🟥	🟩 €€€: units €: packs	🟩	🟩	🟩	€€ 4.6€
8	Company #8	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟩	🟥	Not listed
9	Company #9	🟩	🟩 €€€ 11.6€/unit	🟥	🟥	🟥	🟩	🟥	€ 2.6€
10	Company #10	🟩	🟩 €€€ 14€/unit	🟥	🟥	🟥	🟥	🟥	Not listed
11	Company #11	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
12	Company #12	🟩	🟩 €€€ 12€/unit	🟥	🟥	🟥	🟥	🟥	Not listed
13	Company #13	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
14	Company #14	🟩	🟩 €€€ 34.12€/ unit	🟥	🟥	🟥	🟥	🟥	€ 0€
15	Company #15	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟩	🟥	Not listed
16	Company #16	🟩	🟩 €€€ 25€/unit	🟥	🟥	🟥	🟩	🟥	€€/€€€ 7.2€
17	Company #17	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
18	Company #18	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
19	Company #19	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
20	Company #20	🟩	🟩 €€€ 17.8€/unit	🟥	🟥	🟥	🟥	🟥	€€/€€€ 7.2€
21	Company #21	🟩	🟩 €€€ 32.7€/unit	🟥	🟥	🟥	🟥	🟥	€ 0€
22	Company #22	🟩	🟩 €€€ 22.5€/unit	🟥	🟥	🟥	🟥	🟥	€€ 5.4€
23	Company #23	🟩	🟩 €€€ 14€/unit	🟥	🟥	🟥	🟥	🟥	€€€ 14.4€
24	Company #24	🟩	🟥 B2B Not listed	🟥	🟥	🟥	🟥	🟥	Not listed
25	Company #25	🟩	🟥 B2B Not listed	🟥	🟩 €€ 2€/unit	🟥	🟥	🟩	€/€€ 4€

Figure 19: Table with all the gathered research data about sticker printing companies, created before 10.12.2021

2.3 communication with companies

19 companies have been contacted, and 2 answered some of the questions outlined in the email. Here are the takeaways from these messages.

The first company stated that they started the company with a similar idea, and that is allowing small volume orders for ordinary people. However, they weren't very successful, and the main problems were not just in communication and ordering system but also with work procedures.

The second company is interested in the automatic solution, and they are considering implementing a similar application. They explain that, on average, 45 minutes is spent per customer, and that includes communication, design preparation, administration, printing, cutting, packaging, and shipping. When it comes to low volume orders from ordinary people, the materials often don't have the best design files, their images often don't have sufficient resolution, and text is very small as the printer itself has its own resolution, and in order for a text to be visible it has to has a certain size[7] and also the cutting plotter can be inaccurate[8] and that has to be taken into account when designing.

They shared that the most expensive and the slowest parts of the production were communication with customers, advertisement, and separation of complex cutouts.

At last, they mentioned that they have printers with a width of 16 meters, often print during the night, and the misuse of copyright is the responsibility of the customer.

2.3.1 summary

It can be noted that both companies stated that the communication with customers is very expensive and essential problem to solve in order to provide small volume orders effectively, especially to ordinary people who often don't have the best materials and aren't familiarized with the nuances of sticker printing.

After solving the problem with communication, the production also brings its own set of challenges. Work procedures have to be effectively designed or reworked with automation in mind, as applying such solutions to existing procedures can be problematic.

[7]Given example for text minimal size was 2mm

[8]Given example of inaccuracy was 0.5mm

2.3.2 possible solutions

Following are the possible solutions for some of the problems highlighed by the companies.

- Communication with customers

  When it comes to custom orders, automating the communication with customers is essential in order to be able to support a large number of small volume orders. A web application with an editor can be a great tool to handle the requirements of the customers, paired with a sensibly designed database for managing these orders.

- Insufficient resolution

  After a design file with the insufficient resolution is uploaded to the editor, it can be inspected by the user as editor canvas resolution can be automatically set and thus reflect the resolution of the printer. As the editor reflects the printer resolution, problems with small text should be easily observable. Additionally, users may be prompted with a warning that the file is too small.

- inaccuracy of the cutting plotter

  The editor offers different cutouts for users to choose from, and each cutout can have its own 'safe' area and area, which can be inaccurately cut. Additionally, after designing the file, users could be prompted, for example, with four automatically generated images that show the possible errors and ask them if they are ok with such misalignment or want to edit their designs to solve these errors.

- Materials

  As having an automatic production work procedure that supports different materials per printer can be a difficult problem, it could be better to offer only a limited amount of materials or a singular material to choose from.

2.4 conclusion

If there is a need for an e-shop for stickers, one should be open to being inspired by Redbubble. Their filtration system, presentational images for sizes and different materials, reviews, and the ecosystem for ordinary people to share their designs led to the creation of a database of millions[9] of designs.

[9]source: redbubble.com

On the other hand, if one wants to make an automatic ordering tool for stickers, Stickeryou shouldn't be overlooked. Stickeryou offers a great and fast editor experience with few but sufficient and simple tools for its users to upload and edit their order, including the option for a "die-cut" border. Moreover, Diginate also offers a high-quality editor, however, the speed with which a user can edit their designs is slow and unpleasant. Other than that, Diginate offers the best user experience when it comes to the number of options of available materials and the way of presenting finished orders.

3 Chapter: Application functionality

The core of the application functionality revolves around offering the user high customization options, in this case, for sticker orders.

At the same time, this application offers companies tools for helping them to automate the production processes needed to be able to give users such options for relatively acceptable prices.

The following list describes these functions for the users as well as the company.

A Functionality offered to users

- **Manage account**
  - Login
  - Register
  - Change password (not implemented but desired)
- **Browse stickers**
  - Filter existing stickers by name and tags
  - Browse through similar designs of the sticker on the sticker's page
- **Buy existing sticker designs**
  - Edit sizes of existing stickers
  - Compare the size with familiar object on sticker page
  - Add (remove) existing stickers to (from) cart
  - Being able to add stickers to cart from the gallery page
  - Add (remove) existing stickers to (from) favorites
  - Finalize order
- **Order uploaded stickers**
  - Upload custom image file
  - Picking predefined cutout outlines
  - Defining canvas dimensions in millimeters
  - Picking a material
  - Picking a material-specific style
  - Moving with the uploaded image
  - Painting on the editor (and removing the paint)
  - Picking the color of the paint
  - Erasing contents from uploaded image (and re-erasing)

  - Scaling the image
  - Quantity of canvas stickers to be ordered
  - Add canvas to the cart

B Functionality offered to company

- **Automation**
  - Paid order with design and cutout files ready for printing and cutting in the database
  - Automated stacking of these orders on a page
  - Offering custom sticker printing without the need for communication through the editor
  - Offering resize options for existing designs
- **Web database editor**
  - Managing prices for different materials
  - Easily adding new (removing) available (existing) materials through database
  - Easily adding new (removing) available (existing) cutout shapes through the database
- **Existing stickers**
  - Managing different materials and style options for a particular sticker
  - Being able to present materials and style options of 1 sticker on 1 page
- **Database editor**
  - Easily editing, creating or removing database items with user friendly Django admin application

4 Chapter: Development procedure

This application has been developed in three major parts: research, planning, and implementation.

A Research

- Searching for existing solutions
- Comparing foreign and local companies in different aspects
- Summarizing the best features and planning on top of them
- Reaching out to local companies

B Planning

- Summarizing goals of the application, its functions and strategy in Miro [11]
- Choosing suitable technologies
- Creating frontend design for application pages in Figma [13]

C Implementation

- Creating Django application with PostgresSQL [4] database
- Learning React [3] and creating a React application
- Piecwise implementing parts of the application (frontend + backend API) in a sensible order
- Deploying and testing the application on Heroku [5]

5 Chapter: Frontend general design and implementation

5.1 technologies

Following technologies and services were used in the frontend development:

- React JavaScript library [3]
  - Mui [14] user interface React library with existing component
  - React Axios [15] library for API request handling
  - React Router [16] library for routing and structuring application pages
- Figma [13] design application for frontend design layout for both desktop and mobile devices
- Adobe Photoshop [17] for editing sticker and promo images

5.2 code structure

React codebase is located in React application's *src* folder. In React, React class and functional components[10] are created and nested into each other. This structure and nesting can be represented with a tree graph Figure 20.

Class components in this application are closer to the root and contain functional definitions and variables used by the functional components that it imports. Functional components, in this case, suffice as importable standalone user interface entities. The functionality of these entities is mostly dependable on the variables and functions which are passed as an argument from the component that is importing it.

An example of such a relation can be a Class component representing sticker listing, this component contains variables such as a list of all the stickers it has to display, and in order to display, it imports function component *StickerList* responsible for styling the list and gives it stickers as an argument. Additionally, *StickerList* functional component also import functional component *Sticker* responsible for styling individual stickers which are displayed.

At the root of this tree is *Index.js* file, which contains React *Index* Class with routing URL paths to each separate page of the application, variable and function definitions that are used in multiple pages, which are passed as parameters. One example of such a variable is the variable for storing items a user has in his cart as well as a function that asks the server for an updated version of this cart. Cart stickers variable is being used in every *Navigation* Class instance and is thus passed as an argument.

5.2.1 routing and components

URL routing is handled by React Router library [16] and is defined in the React root's *Index.js* file.

Here is a complete list of all URL paths that the current site uses:

- **Path:** "/", **Page function:** home page with sticker filtering and listing
- **Path:** "/sticker/:sticker_id", **Page function:** separate page for a sticker
- **Path:** "/favorites", **Page function:** listing of stickers user saved as favorites
- **Path:** "/cart", **Page function:** listing of stickers that user has saved in cart
- **Path:** "/cart/continue", **Page function:** page for finalizing orders

[10]reactjs.org/docs/components-and-props.html

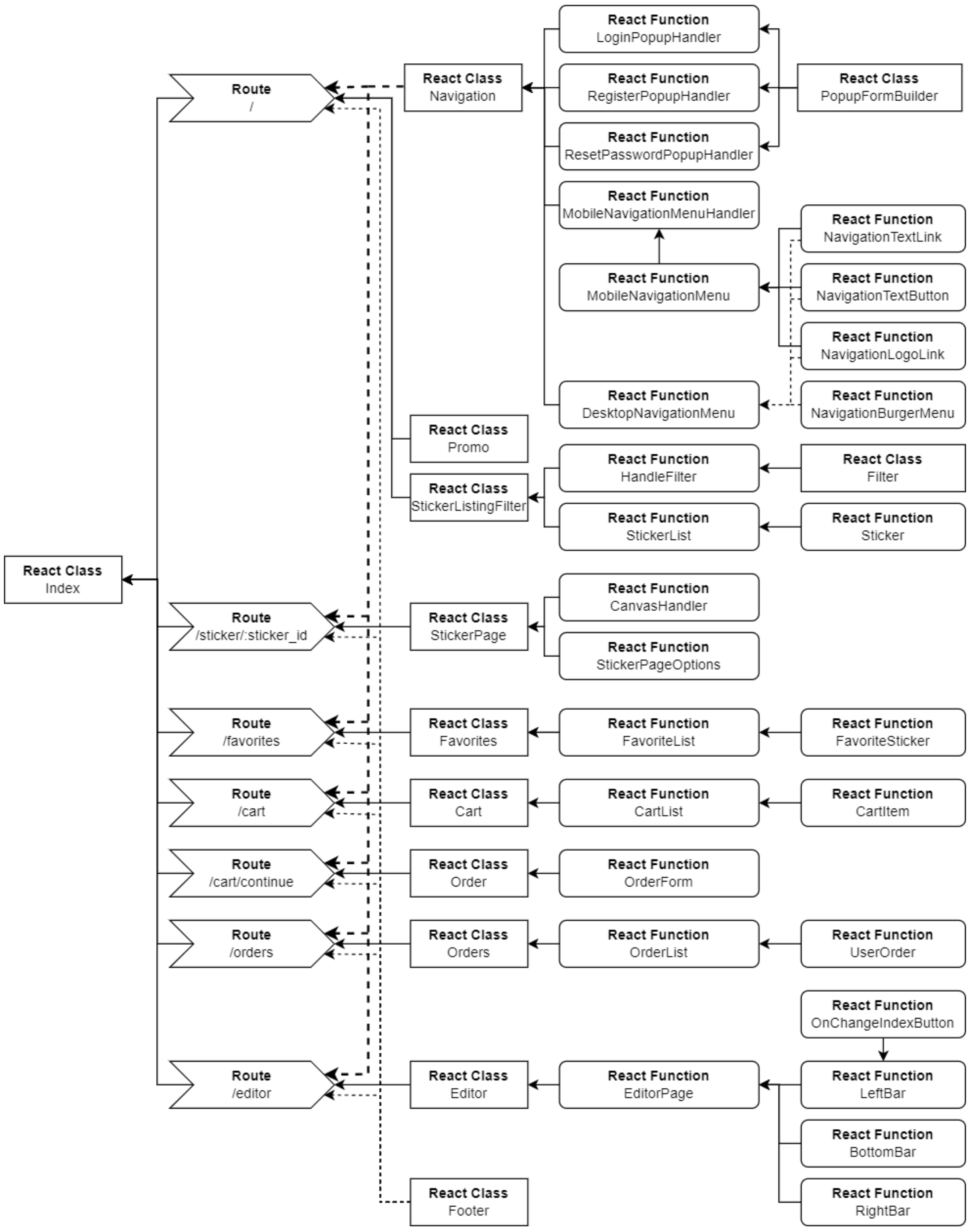
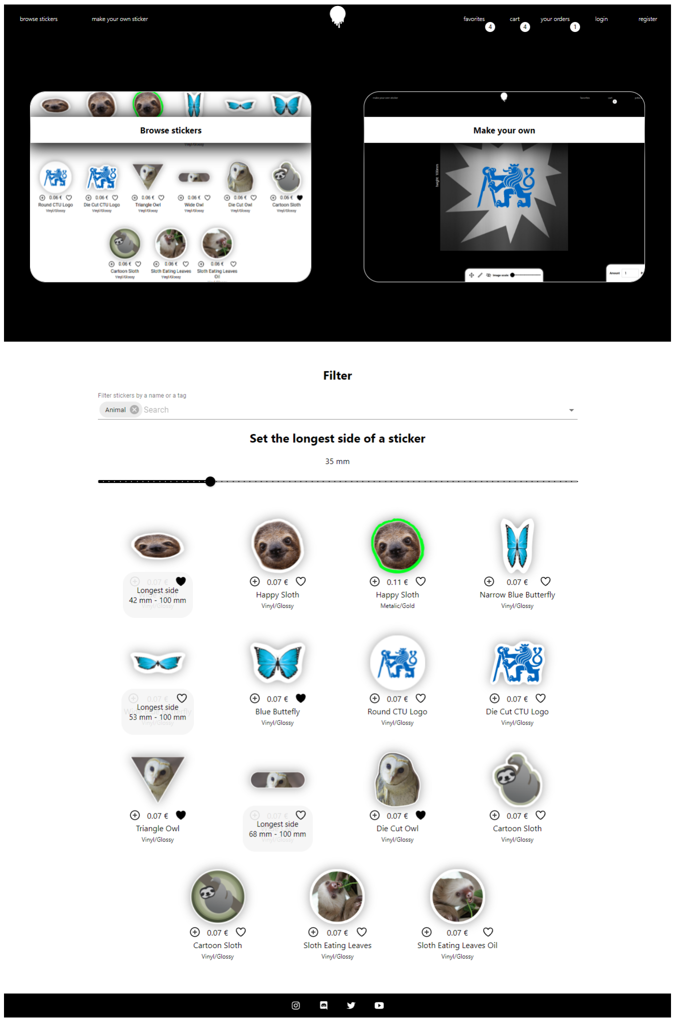


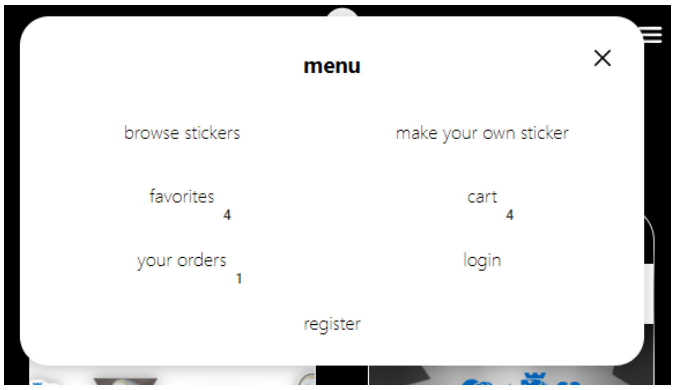
Figure 20: Visualization of React components nesting

5.2.2 visualized

Following images are screenshots of completed and deployed application's user interface.

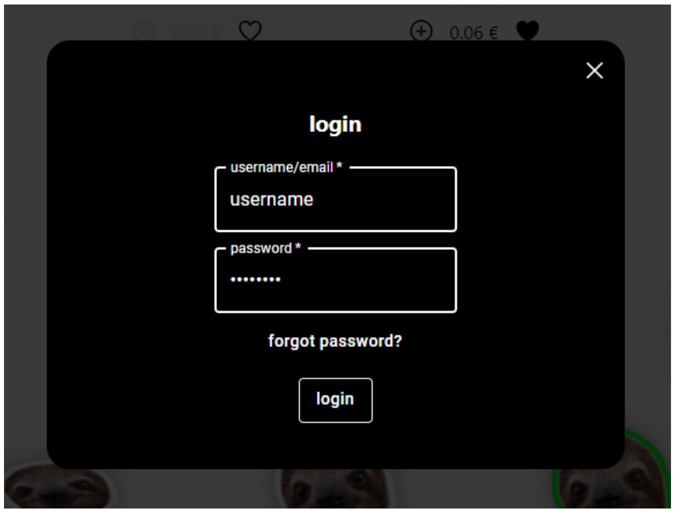


(a) Homepage miniaturized

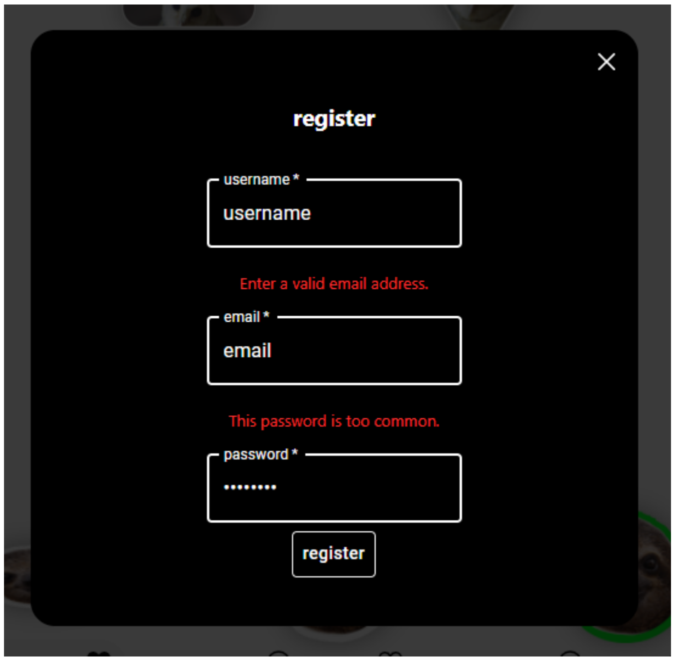


(b) Mobile navigation menu

Figure 21: Homepage and mobile menu



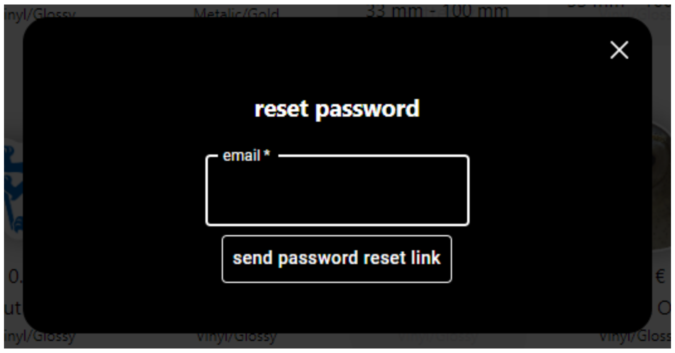
(a) Login popup



Enter a valid email address.

This password is too common.

(b) Register popup



(c) Resetting password popup

Figure 22: Popups for loggin in, registering and resetting password

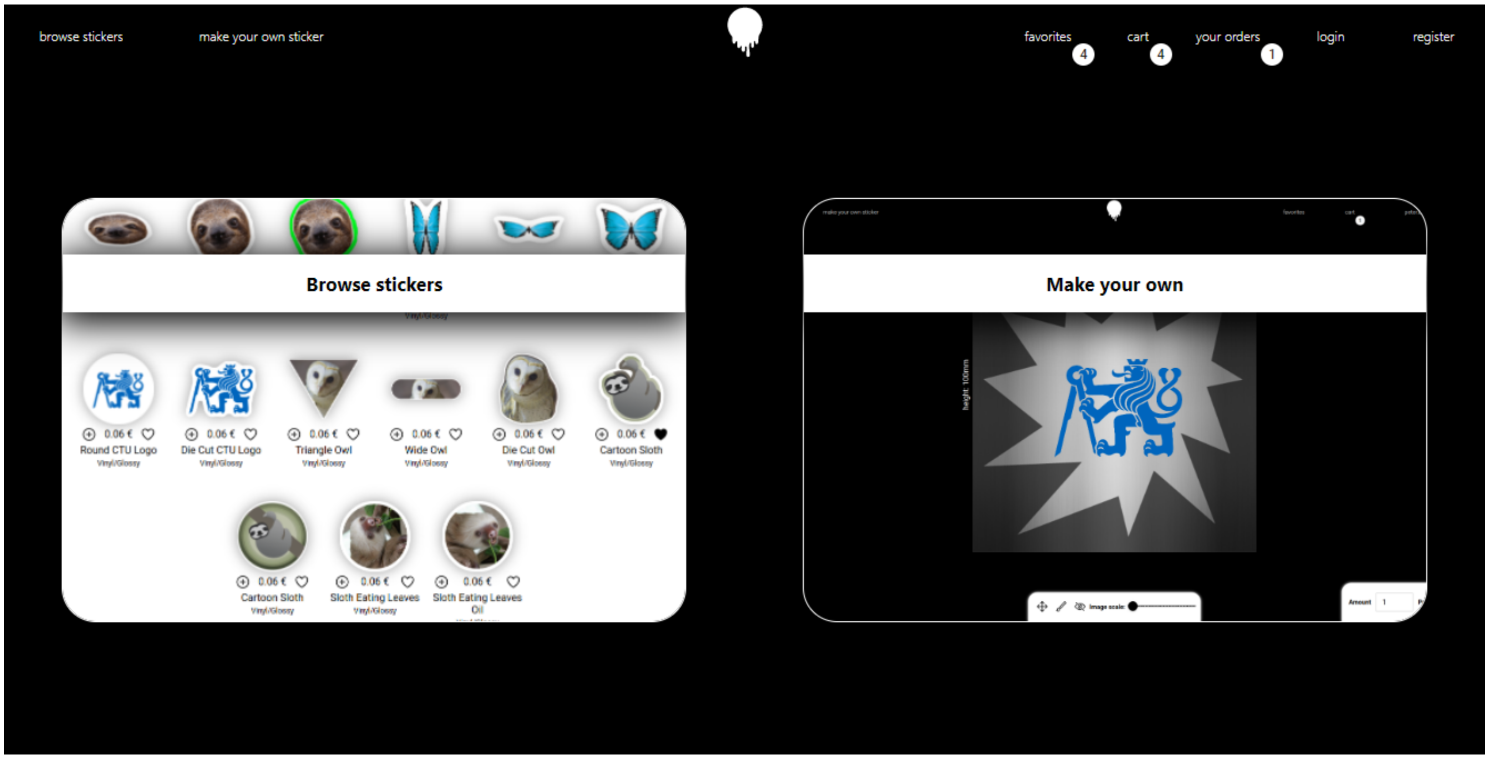
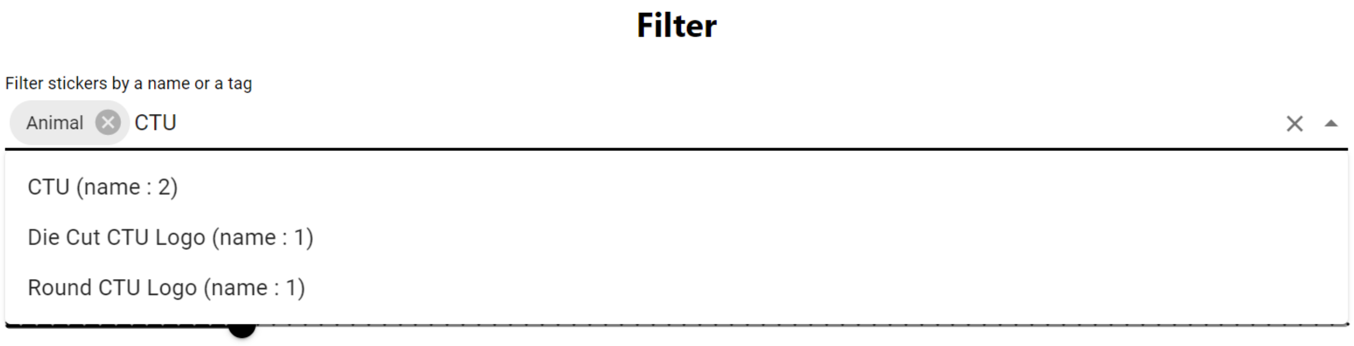


Figure 23: Promo part of the homepage



Filter

Filter stickers by a name or a tag

Animal CTU

CTU (name : 2)

Die Cut CTU Logo (name : 1)

Round CTU Logo (name : 1)

Figure 24: Gallery filter on the homepage

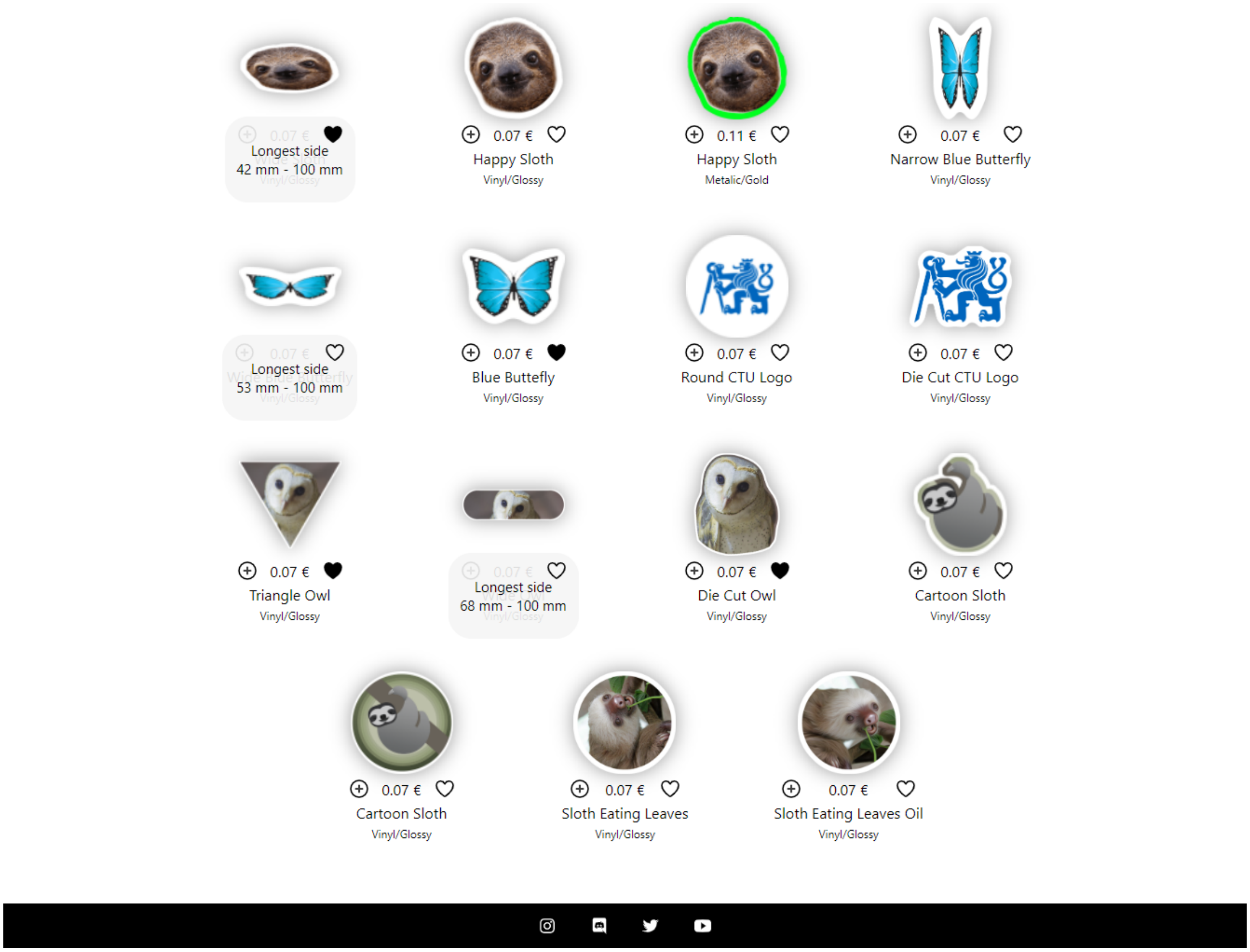
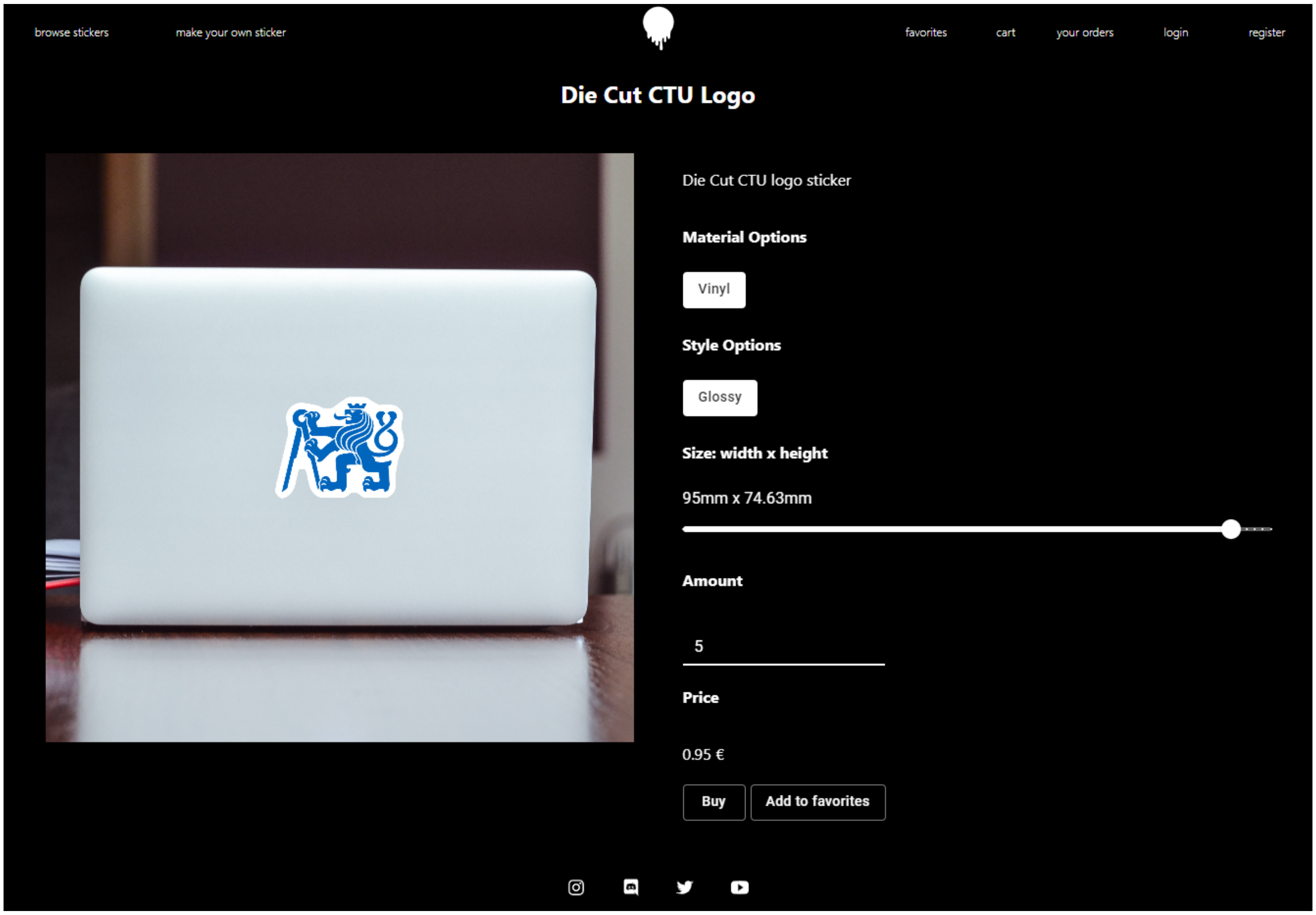


Figure 25: Stickers gallery on the homepage



Die Cut CTU Logo

Die Cut CTU logo sticker

Material Options

Vinyl

Style Options

Glossy

Size: width x height

95mm x 74.63mm

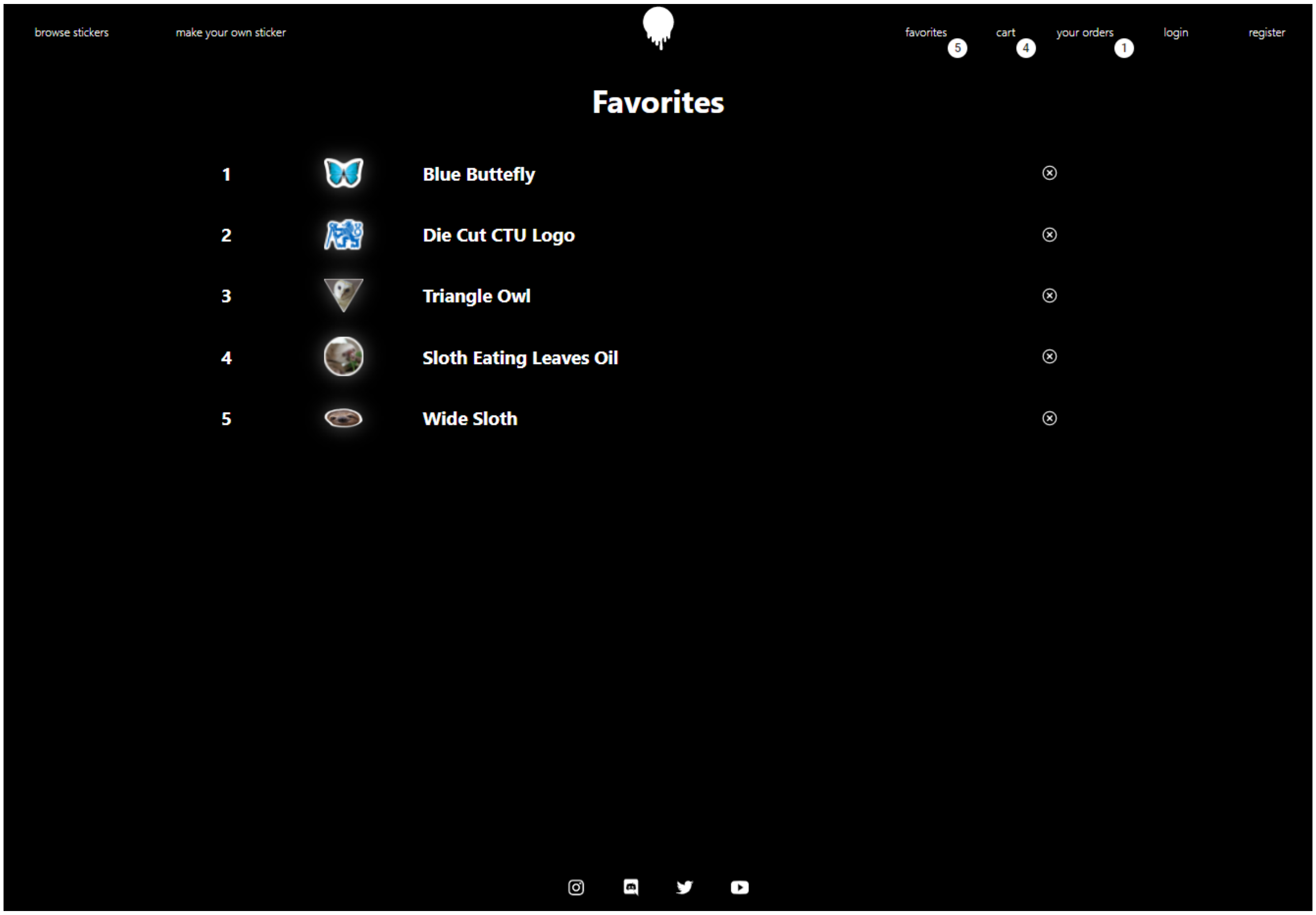
Amount

5

Price

0.95 €

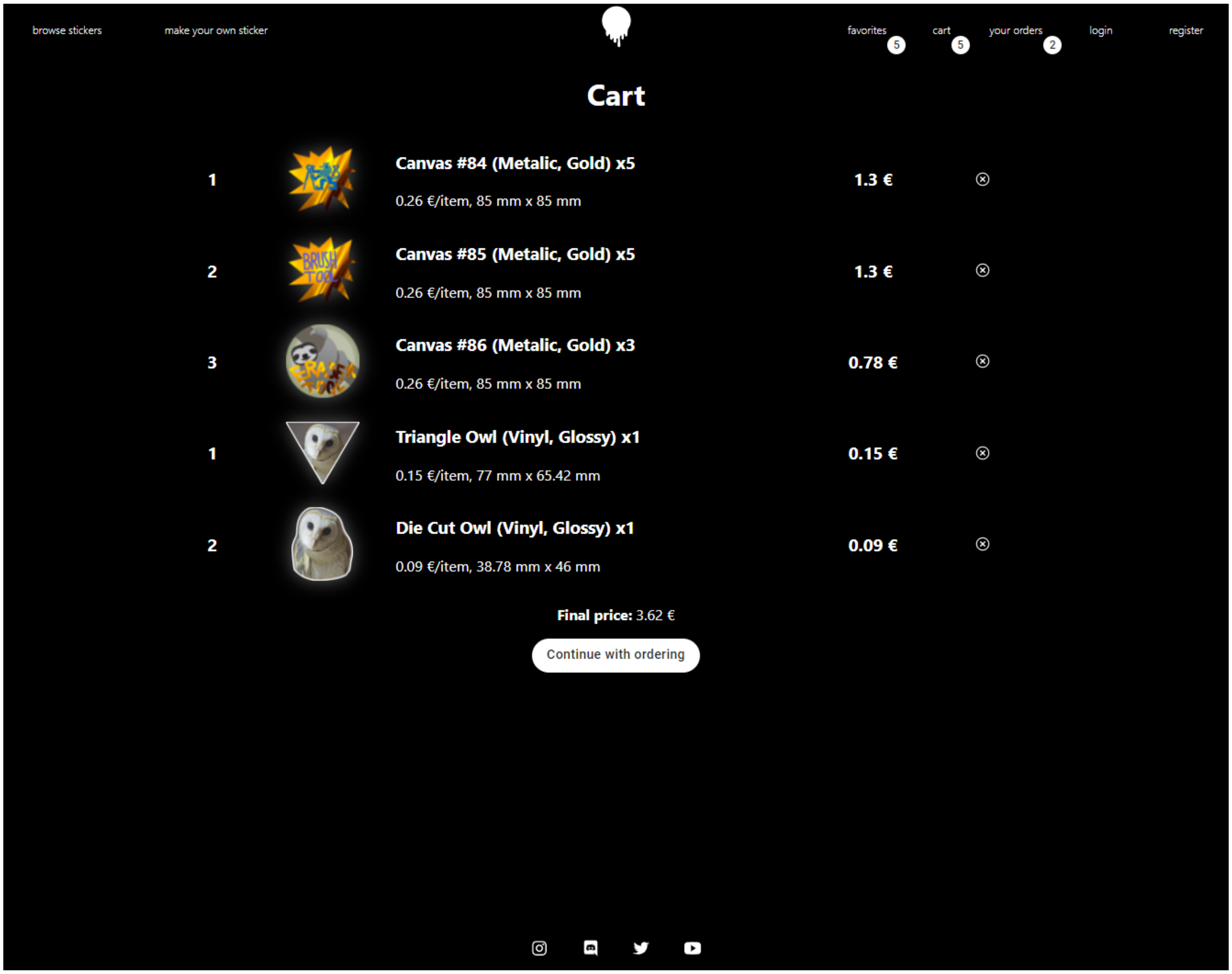
Figure 26: Sticker item page



Favorites

1		Blue Buttefly
2		Die Cut CTU Logo
3		Triangle Owl
4		Sloth Eating Leaves Oil
5		Wide Sloth

Figure 27: Page with user's favorites



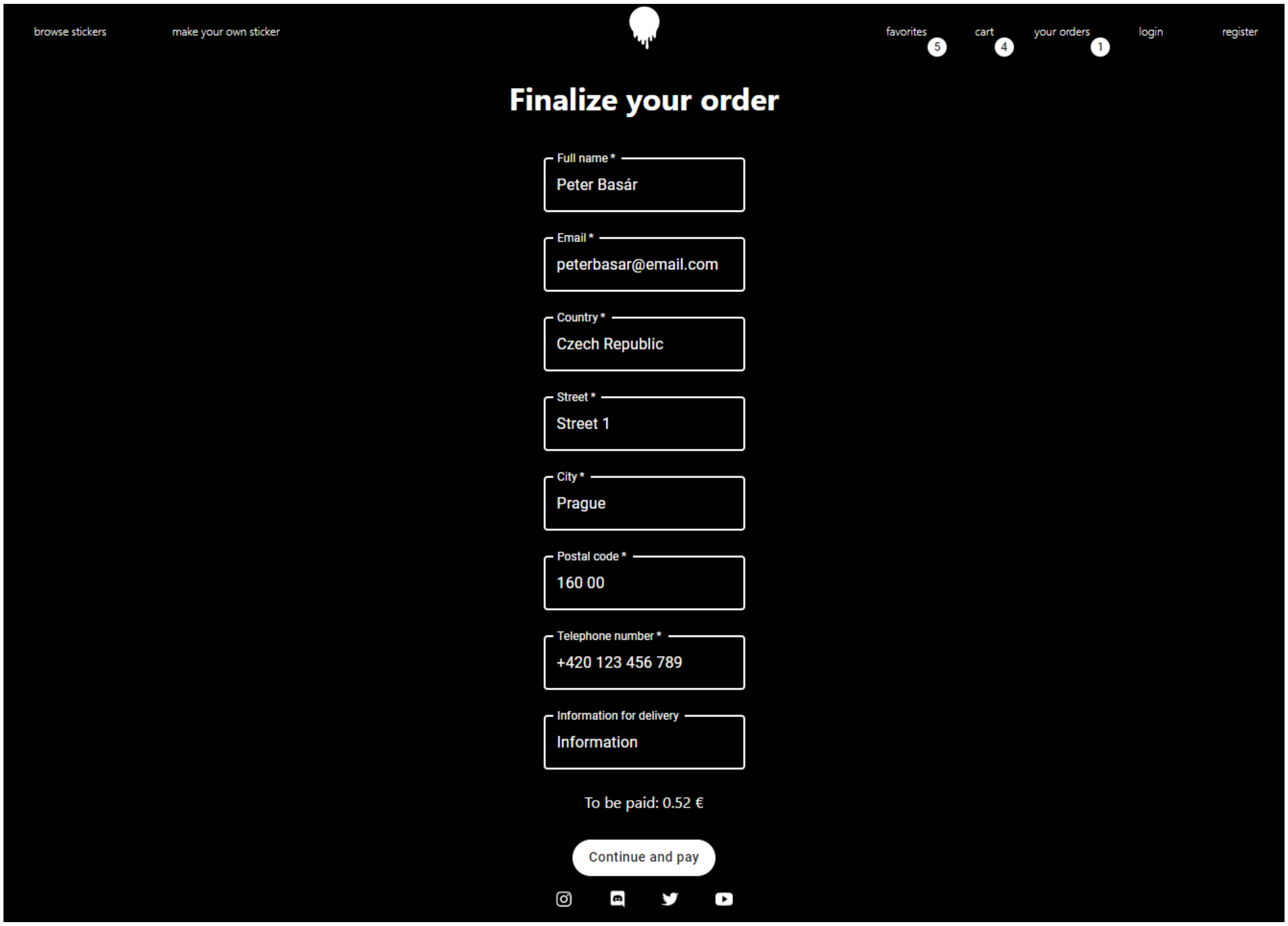
Cart

1	Canvas #84 (Metalic, Gold) x5 0.26 €/item, 85 mm x 85 mm	1.3 €	⊗
2	Canvas #85 (Metalic, Gold) x5 0.26 €/item, 85 mm x 85 mm	1.3 €	⊗
3	Canvas #86 (Metalic, Gold) x3 0.26 €/item, 85 mm x 85 mm	0.78 €	⊗
1	Triangle Owl (Vinyl, Glossy) x1 0.15 €/item, 77 mm x 65.42 mm	0.15 €	⊗
2	Die Cut Owl (Vinyl, Glossy) x1 0.09 €/item, 38.78 mm x 46 mm	0.09 €	⊗

Final price: 3.62 €

Continue with ordering

Figure 28: Page with user's cart items

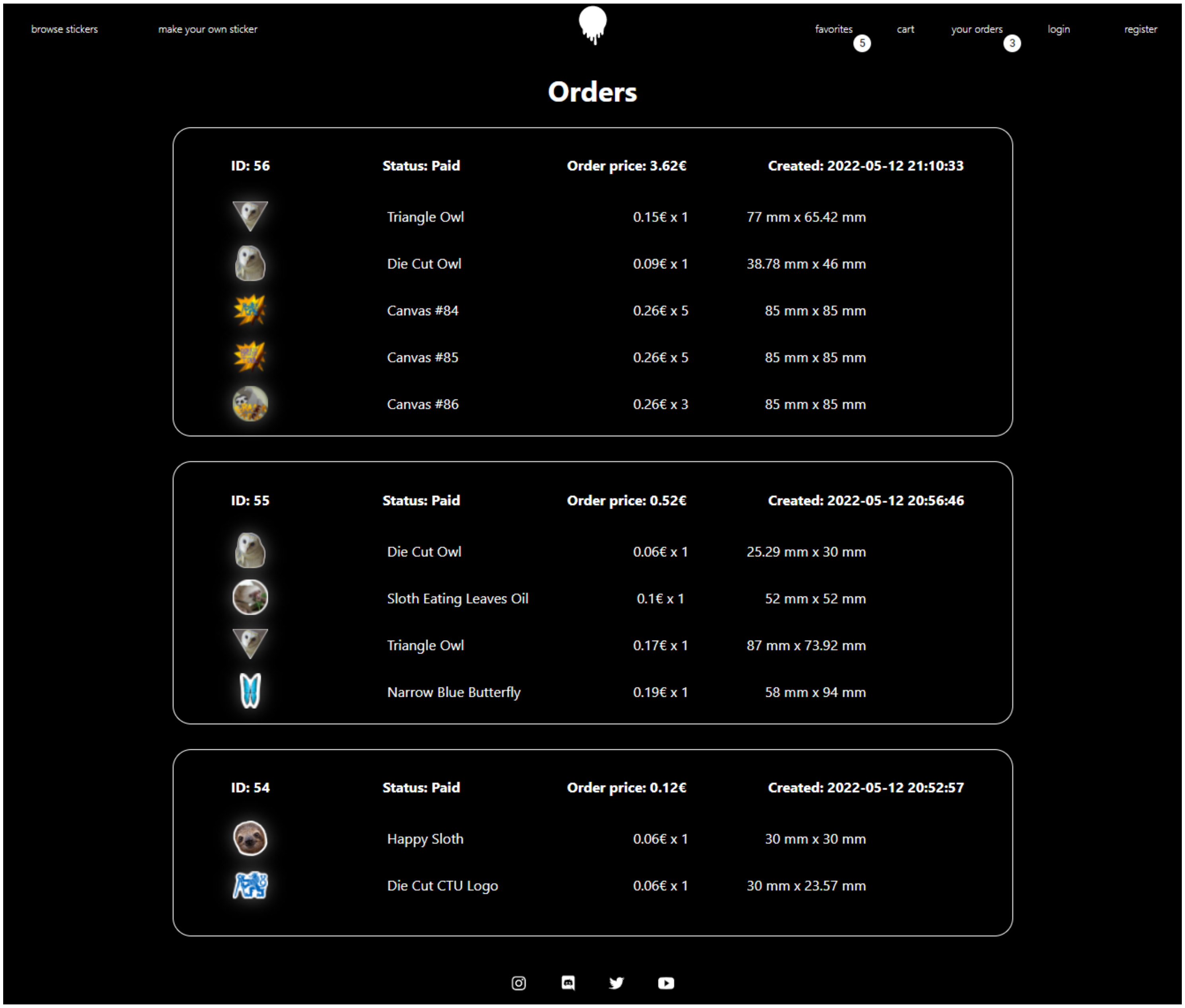

browse stickers
make your own sticker
favorites 5
cart 4
your orders 1
login
register

Finalize your order

Full name *	Peter Basár
Email *	peterbasar@email.com
Country *	Czech Republic
Street *	Street 1
City *	Prague
Postal code *	160 00
Telephone number *	+420 123 456 789
Information for delivery	Information

To be paid: 0.52 €

Figure 29: Page for creating order



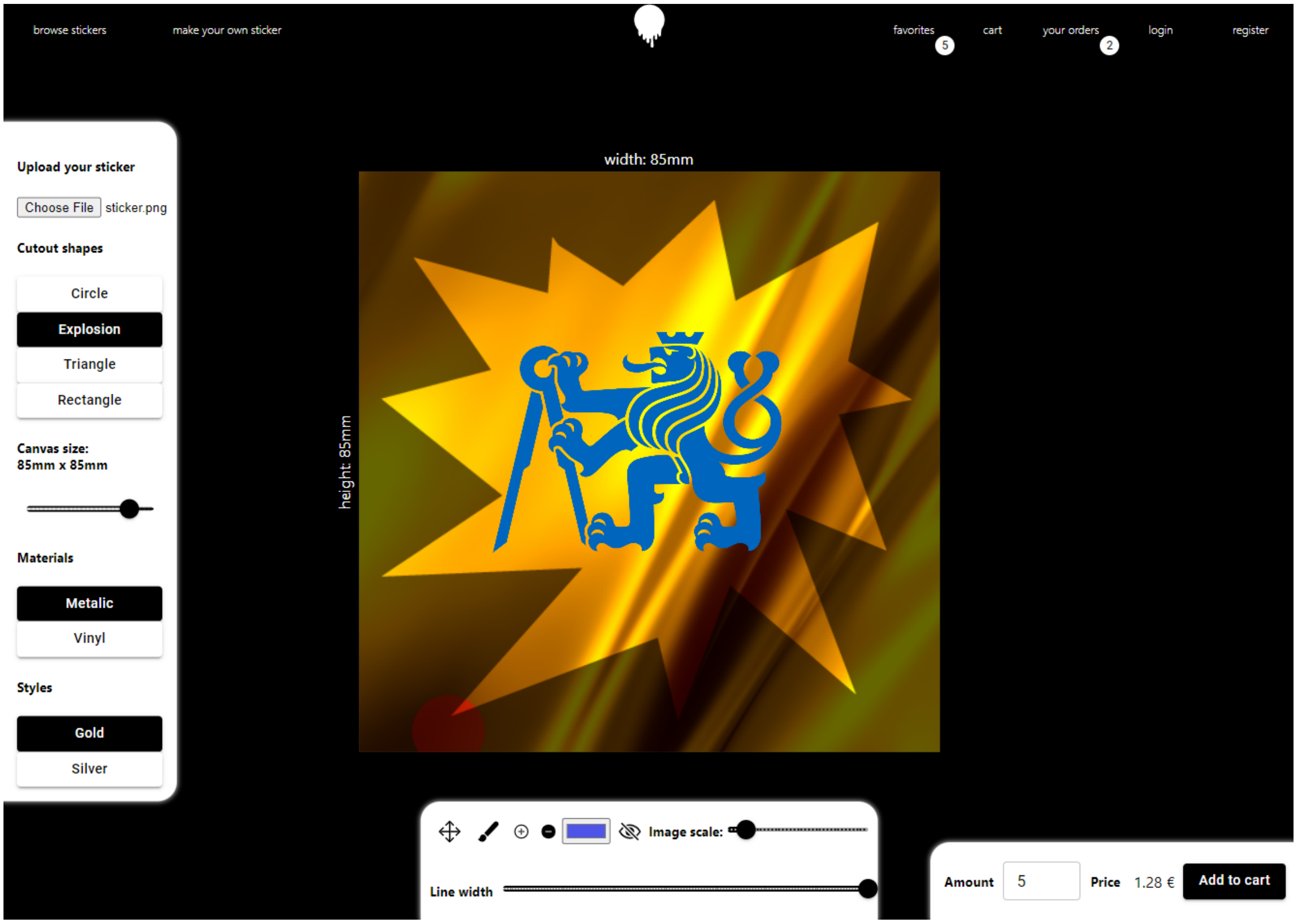
Orders

ID: 56	Status: Paid	Order price: 3.62€	Created: 2022-05-12 21:10:33
	Triangle Owl	0.15€ x 1	77 mm x 65.42 mm
	Die Cut Owl	0.09€ x 1	38.78 mm x 46 mm
	Canvas #84	0.26€ x 5	85 mm x 85 mm
	Canvas #85	0.26€ x 5	85 mm x 85 mm
	Canvas #86	0.26€ x 3	85 mm x 85 mm

ID: 55	Status: Paid	Order price: 0.52€	Created: 2022-05-12 20:56:46
	Die Cut Owl	0.06€ x 1	25.29 mm x 30 mm
	Sloth Eating Leaves Oil	0.1€ x 1	52 mm x 52 mm
	Triangle Owl	0.17€ x 1	87 mm x 73.92 mm
	Narrow Blue Butterfly	0.19€ x 1	58 mm x 94 mm

ID: 54	Status: Paid	Order price: 0.12€	Created: 2022-05-12 20:52:57
	Happy Sloth	0.06€ x 1	30 mm x 30 mm
	Die Cut CTU Logo	0.06€ x 1	30 mm x 23.57 mm

Figure 30: Page with user's orders information



sticker.png

Canvas size: 85mm x 85mm

width: 85mm

height: 85mm

Amount 5

Price 1.28 €

Figure 31: Editor interface

5.3 functionality

This application contains many functionalities on separate and often the same pages. In order to present them in an easy-to-understand way, they will be firstly individually described and then mapped in a graph under a page that uses it.

5.3.1 local storage

Local storage is being used so that variables that are widely used within the application are easily accessible. Among these variables are authentication tokens, username, and a variable representing if the user is temporary or authenticated (changing the variable doesn't authenticate the user, its purpose is visual).

Variable names are defined in variable *src/ GlobalConstants.js / LOCAL_STORAGE*.

5.3.2 navigation and authentication

Through navigation menu, the user is being offered different options to traverse through the different pages of the application or execute specific on-click functions.

These are current clicable options for user in the navigation menu:

- **"browse stickers"** button that redirects users to home page
- **"create your own stickers"** button that redirects users to the editor
- Clickable **logo** that redirects users to home page
- **"favorites"** button that redirects users to page with stickers saved as favorites
- **"cart"** button that redirect user to a page with cart items and order finalization
- **"login"** button that opens a floating dialog box for logging in
- **"logout"** button that logs out a logged in user
- **"register"** button that opens a floating dialog box for registering
- clickable **hanburger** icon which opens up mobile menu if user is accessing this application on mobile

5.3.2.1 desktop and mobile navigation

The navigation menu has both desktop and mobile versions. Both are React components, and both are imported. Depending on the width of the screen, if a screen is very narrow, the mobile version will be active, while the desktop version will disable all of its clickable buttons except the logo and hamburger icon (which are now visible and active). When in desktop mode hamburger icon and mobile navigation is hidden, and clickable desktop menu buttons are active.

`images/kapitola_5/functionality/desktop_moblie_navigatio`

Figure 32: Desktop (top) and mobile navigation (bottom two) menu

5.3.2.2 number badges for cart and favorites

After a user adds a sticker to either favorites or a cart, the application updates its lists for cart and favorite stickers with an API call and also updates numbering in the navigation menu that symbolizes the total count of items saved to favorites and to cart. Badges have been implemented with Mui Badge[11] component.

[11]mui.com/material-ui/react-badge/

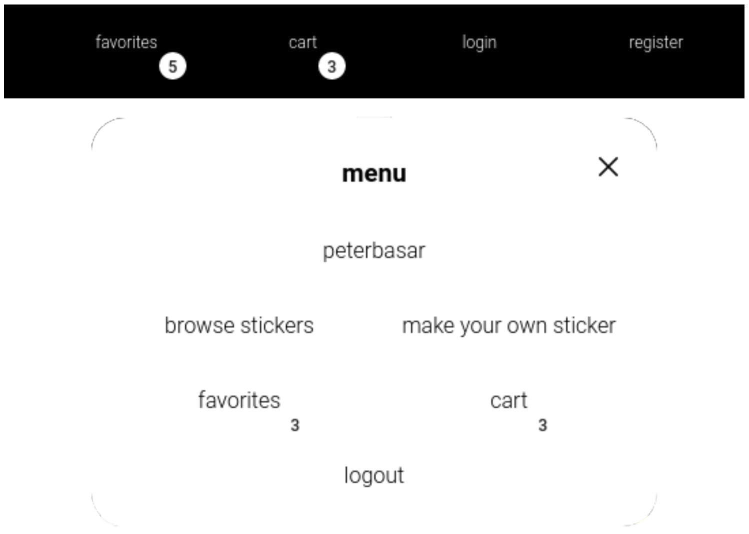


Figure 33: Number badges for favorite and cart item count

5.3.2.3 JWT authentication Application authentication utilizes JWT technology. On API calls that require users to be authenticated (user's data-related calls such as managing favorites, cart, orders, ...), it passes the access token to a header as a Bearer token argument in format *Authorization: "Bearer <access token>"*.

In order to utilize JWT authentication and at the same time offer services such as managing favorites and stickers in the cart even to users who did not register and log in on a visit, the application searches local storage for access and refresh tokens and verifies them Figure 51. If no valid access token is available, frontend requests for a new temporary user (which is created in the database) and receives appropriate tokens. At the same time, local storage variable *user/auth/is_temp_user* is being set to 1.

5.3.2.4 logging in After being prompted with the login dialog window, the user can enter login information, either username or email and password specified during registration, and click on the confirmation button. After clicking on a button, React uses variables that "mirrored"[12] form text inputs and makes login API call (see 6.2.1 for API definitions).

[12]React Class component PopupFormBuilder, responsible for dialog windows for login, registering, and changing password, contains variables which are copying contents of form text inputs on any change in those inputs

On successful response, the user receives access and refresh tokens. These tokens are then saved in local storage, and a variable representing if the current user is temporary or not[13] is set to 0.

5.3.2.5 logging out Logging out is available only to logged-in users and can be accomplished by pressing the logout button in the navigation menu.

After user presses the button, all the tokens from local storage are deleted, local storage variable (Section 5.3.1) *user/auth/is_temp_user* is set to 1, and web application is refreshed (triggering on visit request for temporary tokens Figure 51).

5.3.2.6 registering After being prompted with the registration dialog box, the user has to enter the username, email, and password.

After confirming the registration, the appropriate API call is being made (see 6.2.1 for API definitions).

On successful response, the registration dialog box is turned off, the login dialog box turned on, and the user can now log in.

5.3.2.7 changing password One can access the dialog box for changing the password from the login dialog box, which contains a "forgot password" button. This button closes the login dialog box and opens up the change password dialog box.

The functionality of changing the password is not implemented due to the prioritization of other more essential features.

5.3.3 color managment

The vast majority of colors displayed in the application are being imported from the *GlobalConstants.js* file located in the React's root *src* directory.

Colors are defined in *COLORS* objects that map color names to color hexadecimal representation.

5.3.4 handling responsivness

5.3.4.1 grid system Mui grid system[14] is being used to define the layout of the user interface.

[13]All users who visit the application are temporary and have temporary access tokens

[14]mui.com/material-ui/react-grid/

Dynamic behavior definition is being done by specifying the widths of a *Grid* component parameters (xs, sm, md, lg, xl) as well as other parameters specifying its behavior and alignment. Additionally, the user has to work with React syntax and CSS (cascading style sheets) to match the desired behavior.

5.3.5 filtering system

The filtering system is available on the home page over the sticker listing and offers filtering by custom name, name of particular sticker, tags, and combinations of these. By default, there are no filtration parameters set.

5.3.5.1 tags and names The functionality of this filtration system works by providing the query parameters in the get request URL. These parameters are obtained from Mui's *Autocomplete*[15] component, which, if provided with options, offers a user an interface for choosing from these options and searching through these options with text input.

However, *Autocomplete* component requires that we give it options that it can work with. And so it has to repeatedly update its options with an appropriate API call (see Section 6.2.1).

Updating options is being done specifically with function *src/ Components/ App_StickerListing/ App_StickerListingFilter.js/ initialSearchBoxFetch(value)*. This function takes text input as an argument, and if such text contains less than three characters, no filtration is being done in the backend, and the backend returns the most popular tags. If the text contains more than three characters, the backend returns tags, sticker names, and input text as one of the options, as well as the count of how many stickers contains such a string of characters.

This update is being called on page visit, on *Autocomplete* text input change.

[15] mui.com/material-ui/react-autocomplete/

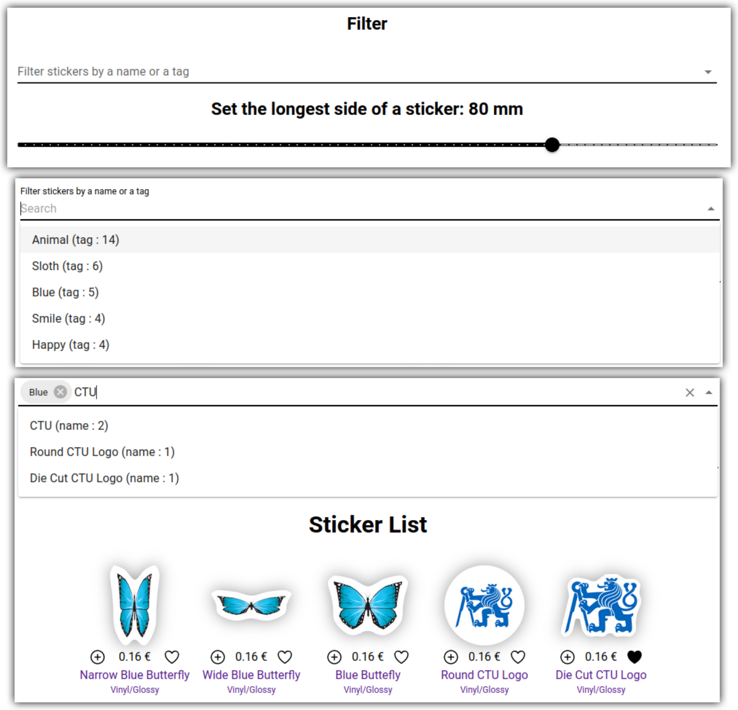


Figure 34: Three examples of the same filter in different states

5.3.5.2 sticker dimensions In order to give users the ability to add stickers to a cart from the listing and customize its size, there has to be some sort of slider or input field, and that's where the slider for picking the longest sticker dimension comes in.

This slider gives users the ability to pick the sticker size they are looking for right from the menu. Along with informing the server which size we want, we also recalculate the prices of the presented stickers.

Additionally, coupled with names and tag filtering, this "longest sticker dimension" is being sent to the server requesting stickers to present.

5.3.6 product page

The main purpose of the product page is to present the selected sticker, its materials, and price and give users the option to save and buy the sticker as well as pick the amount.

5.3.6.1 sticker presented on familiar item Presenting is being done on a familiar item, in this case, a laptop, with a real scale.

In order to be able to know how many pixels such sticker should have, we have to know the width and height of the presented laptop image and the real-life scale of such image (how many millimeters is one pixel).

To further improve presentation, multiple familiar items should be used on different sticker sizes.

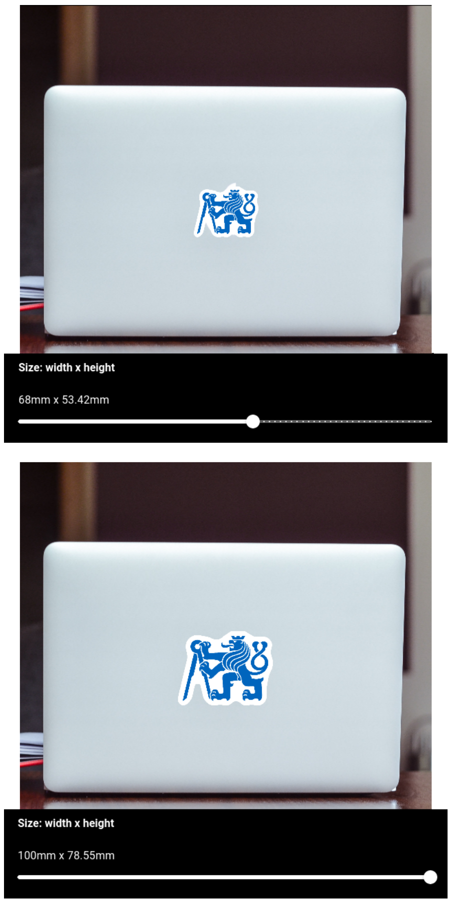


Figure 35: Sticker presentational image for 2 different sizes

5.3.6.2 choosing material and style Each sticker has some id that is specified in the URL of the product page. In order to present the sticker, its materials, and styles, we have to make an appropriate API call by specifying the sticker id as an argument.

In the received data, besides general sticker information, there are nested JavaScript objects such that sticker material options contain multiple sticker style options. This structure makes it easy to implement material and style selection.

Basically, at the initial visit, we initialize and keep track of the indexes for material and style options (these are lists in the received object). And after the user presses other generated material of style, these indexes are updated. Consequently, after ordering or adding the sticker to favorites, we use these indexes to find the style id, and only this id is being used to uniquely identify sticker, material, and style options. This is possible as a result of the database structure.

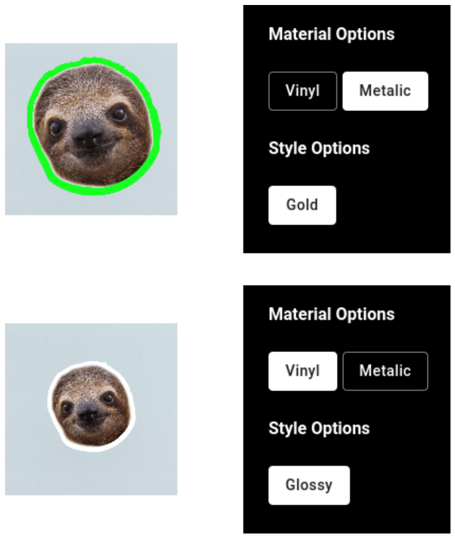


Figure 36: Presented sticker change after choosing different material

5.3.6.3 amount picker Implemented with Mui's *TextField* component that allows single incrementing changes, and integers above value 1.

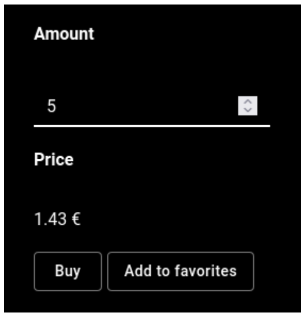


Figure 37: Filed for choosing sticker amount on product page

5.3.6.4 price calculation Price calculation is a very simple linear function that takes in the price parameter of an individual sticker style specified in the database as *price_per_square_mm* and multiplied by the current sticker's longest side.

Such price calculation is defined in function *src/ Components/ App_StickerListing/ App_StickerListingFilter.js/ recalculateStickerPrices(longestSidePick)* and */src/ Components/ App_StickerPage/ App_StickerPage.js / updateStickerPrice()*.

5.3.7 handling favorite stickers

All the data about which stickers are saved as favorites is saved in the React root *Index* class and passed down to other components as well as functions defining the addition and removal of such stickers from the list.

All that is needed to add (remove) the sticker to (from) favorites is a parameter specifying sticker style option id (unique sticker design identifier) and making an appropriate API call (see Section 6.2.1). After these actions, the frontend also automatically requests an update of favorited stickers list with an API call.

This function is available in sticker listing as well as product page with visual indications of which sticker is already added to favorites.

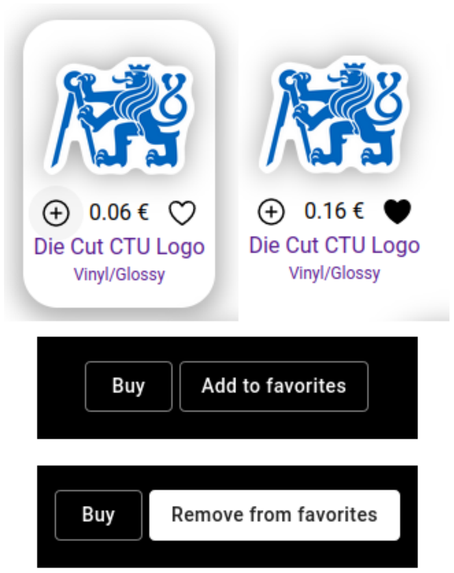


Figure 38: Examples of buttons that manage favorites

5.3.7.1 listing favorite stickers Upon visiting URL */favorites*, the user is presented with all the stickers that are currently his favorites and has options to visit those stickers by clicking on them or removing them from the list.

All the data necessary to list these stickers are being obtained by an API call to the appropriate url (see Section 6.2.1).

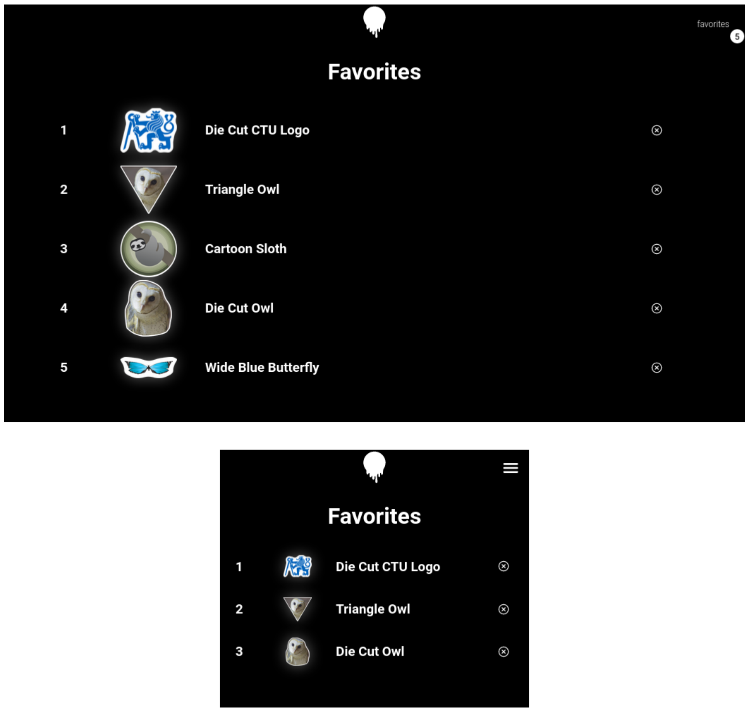


Figure 39: Cropped part of the favorites page for desktop and mobile device

5.3.8 handling cart stickers

Similar to Section 5.3.7 that describes managing of favorite stickers, the cart is almost exactly the same structure but with a different name and just a slightly edited database structure (see Figure 47).

Same as with favorites, appropriate API calls have to be made that uniquely specify the sticker (sticker style option id). Additionally, information regarding quantity and size has to be provided.

Users can add (remove) stickers to (from) their carts on the listing and product page.

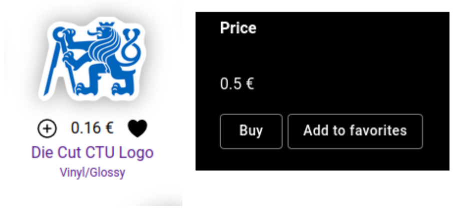
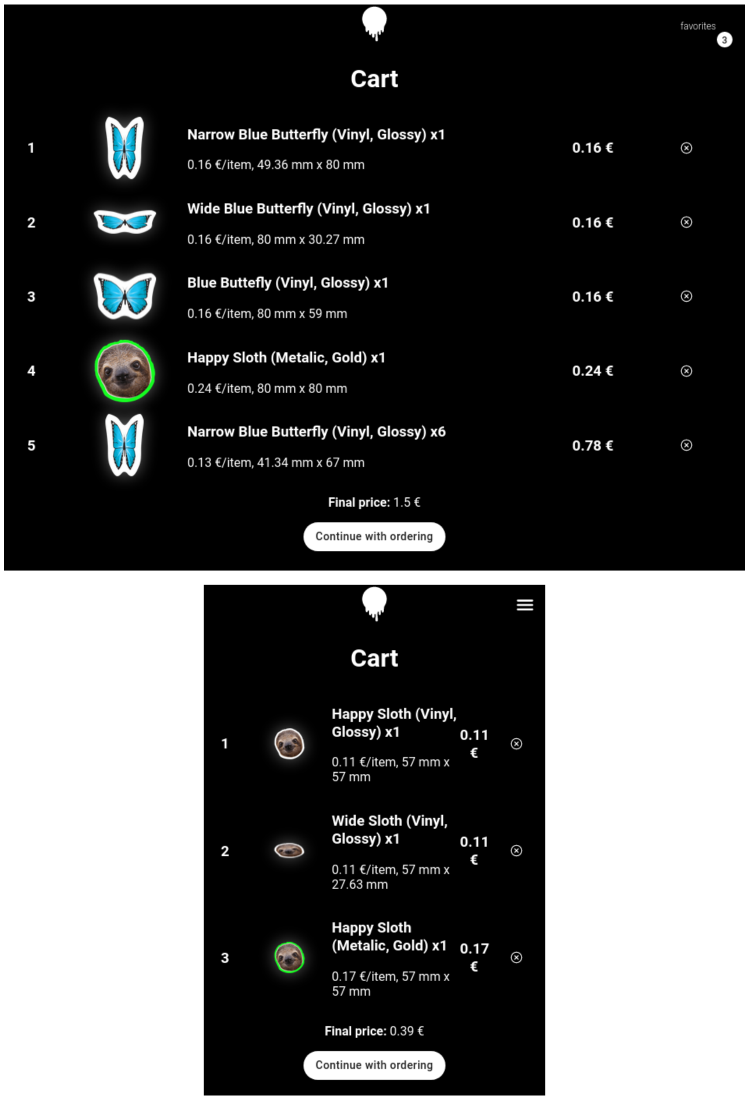


Figure 40: Examples of buttons used for adding stikers to cart

5.3.8.1 listing stickers in cart Similarly to **??** 5.3.7.1, in order to list stickers, we have to ask the server for the specific user's information.

This list can be visited on a specific URL */cart*. Moreover, information about dimensions, price of the individual stickers as well as the whole value of the cart is being presented.

Whilst in the cart, users can remove their cart items.


favorites 3

Cart

1		Narrow Blue Butterfly (Vinyl, Glossy) x1 0.16 €/item, 49.36 mm x 80 mm	0.16 €	⊗
2		Wide Blue Butterfly (Vinyl, Glossy) x1 0.16 €/item, 80 mm x 30.27 mm	0.16 €	⊗
3		Blue Buttefly (Vinyl, Glossy) x1 0.16 €/item, 80 mm x 59 mm	0.16 €	⊗
4		Happy Sloth (Metalic, Gold) x1 0.24 €/item, 80 mm x 80 mm	0.24 €	⊗
5		Narrow Blue Butterfly (Vinyl, Glossy) x6 0.13 €/item, 41.34 mm x 67 mm	0.78 €	⊗

Final price: 1.5 €

Continue with ordering

Cart

1		Happy Sloth (Vinyl, Glossy) x1 0.11 €/item, 57 mm x 57 mm	0.11 €	⊗
2		Wide Sloth (Vinyl, Glossy) x1 0.11 €/item, 57 mm x 27.63 mm	0.11 €	⊗
3		Happy Sloth (Metalic, Gold) x1 0.17 €/item, 57 mm x 57 mm	0.17 €	⊗

Final price: 0.39 €

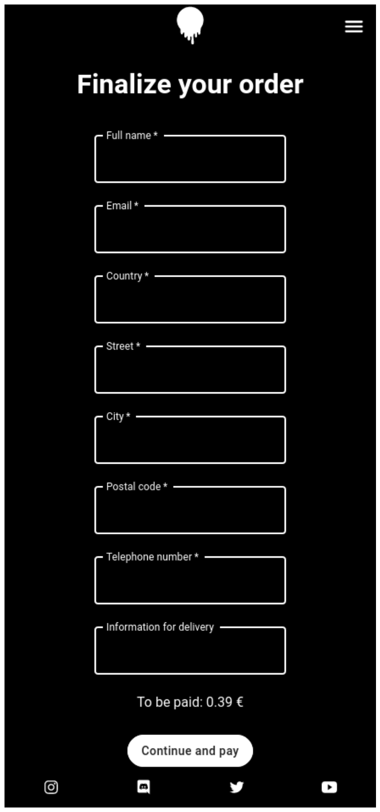
Continue with ordering

Figure 41: Cropped part of the cart page for desktop and mobile device

5.3.9 making an order

After the user is satisfied with what is saved in the cart, and the button to continue with the order in the cart page is pressed, he is redirected to the page for order finalization on URL */cart/continue.*

On this page, the user has to specify basic contact information and finalize the order. On successful order finalization, a new order is created in the database with all the items currently saved in the cart, and the user's cart is emptied.


Finalize your order

Full name *

Email *

Country *

Street *

City *

Postal code *

Telephone number *

Information for delivery

To be paid: 0.39 €

Continue and pay

Figure 42: Order finalization page for mobile device

5.3.10 editor

Editor, in this case, is an interactive 2D space on a computer screen that offers different useful tools for visual editing. After the user of the editor is finished, the editor data is sent to and processed by the application backend.

The architecture of the editor is heavily employing a layering system with canvas HTML elements[21]. Implementation of the editor can be found in the React codebase [20].

The current implementation of the editor is solely designed for desktop devices and offers this functionality:

- Upload custom image file
- Picking predefined cutout outlines
- Defining canvas dimensions in millimeters

- Picking a material
- Picking a material-specific style
- Moving with the uploaded image
- Painting on the editor (and removing the paint)
- Picking the color of the paint
- Erasing contents from uploaded image (and re-erasing)
- Scaling the image
- Quantity of canvas stickers to be ordered
- Add canvas to the cart

5.3.10.1 tool managment The editor contains the following tools:

- Move tool
- Paint tool
- De-paint tool (removing paint)
- Eraser tool
- De-eraser tool (restoring erased area)

Each one of the tools is represented by an index 0-4. Additionally, each tool also has tool states. These states are mostly describing the current state of the tool, meaning that if the user presses the button, the currently selected tool is in active mode (on mouse move, it does its predefined function such as move image or paint) or inactive mode (the tool doesn't do anything). These tools utilize mouse event listeners.

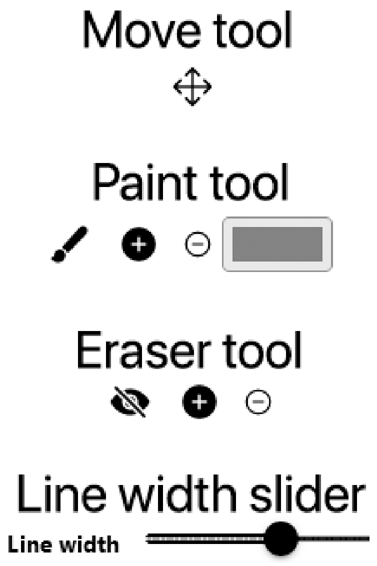


Figure 43: editor tools and their width adjustment slider, plus and minus indicate current tool option (either adding to a layer or removing from layer)

5.3.10.2 layering system The paint tool, and eraser tool each have their own hidden (not visible on screen) canvas layers implemented using HTML canvas element [21]. The main difference between the two is that the eraser tool is scaling and moving with the image (if we move the image, we move the eraser tool layer). Such relationship can be observed on Figure 44.

Only one layer is visible, and that is presenting layer. Presenting layer is regularly rerendered after the user makes an action that changes the visual appearance of this layer. The rendering process is a series of following actions. Use an eraser mask to remove parts of the uploaded image, place the image on the presenting layer, and place the paint layer on presenting layer.

A cutout image is used to establish the initial dimensions of the whole canvas. Changing the cutout image automatically changes canvas width and height. The cutout image is not being used in rendering but is just being placed on top of the presenting layer.

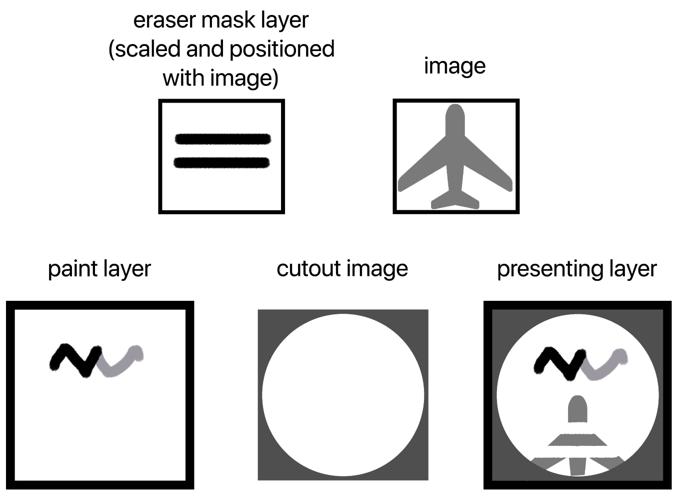


Figure 44: Layering system illustration

6 Chapter: Backend general design and implementation

6.1 technologies

Following technologies and services were used in the backend development:

- Web framework Django[1] for server-side web development.
- PostgresSQL[4] database system for storing the Django model instances.
- AWS S3 storage solution[6] for storing images of stickers and other image media.
- Miro[11] tool for sketching ER diagrams for database and for laying out the early application design.

6.2 code structure

Django project source code[20] is being stored in a separate folder *web_app_django*. Main Django application[16] has the same name as the project and additional Django application is *app_sticker_listing*.

Django applications can each hold the model[17] definitions. and views[18] file with response definitions on API requests. Routing is handled by URL dispatcher which links the user-specified URL[19] patterns to user-created views functions[20]. In this project, all URL patterns are specified in *web_app_django*'s application URL file.

The following list summarizes each application by its use case and what definitions it contains.

A Application: web_app_django

- **Functions**
  - This application handles all the user-related data such as authentication.
- **Models**

[16]docs.djangoproject.com/en/4.0/ref/applications/
[17]docs.djangoproject.com/en/4.0/topics/db/models/
[18]docs.djangoproject.com/en/4.0/topics/http/views/
[19]docs.djangoproject.com/en/4.0/topics/http/urls/
[20]user-created function defined in application's views.py file

  - User()[21]
- **Views**
  - GetTemporaryUserTokens(APIView)[22]
  - LoginView(generic.CreateAPIView)
  - RegisterView(generic.CreateAPIView)

B Application: app_sticker_listing

- **Functions**
  - This application handles sticker-related data such as listing, filtering system, and sticker data requests.
- **Models**
  - SizeLimit()
  - Material()
  - StickerMaterial(Material)[23]
  - Style()
  - StickerStyle(Style)
  - Tag()
  - StickerTag(Tag)
  - Gallery()
  - GalleryItem()
  - Sticker(GalleryItem)
  - MaterialOption()
  - StickerMaterialOption(MaterialOption)
  - StyleOption()
  - StickerStyleOption(StyleOption)
- **Views**
  - parseStickerObjects(max_side_length, stickers)
  - parseFilterOptions(filterTags, filterNames, text="")
  - parseStickerData(sticker)
  - parseQuery(request)
  - filterStickersByTags(tags,objects=None)
  - filterStickersByNames(names,objects=None)
  - StickerList(APIView)
  - GetFilterOptions(APIView)
  - StickerPage(APIView)

[21]Empty argument represents inheriting from Django default model class django.db.models.Model

[22]APIView means that this function is being called as an API view function by URL handler

[23]Argument represents parent class which child class is inheriting from

C Application: app_cart_favorites

- **Functions**
  - This application handles data related to sticker favorite listing and cart managment.
- **Models**
  - Cart()
  - CartItem()
  - StickerCartItem(CartItem)
  - Favorites()
  - FavoriteItem()
  - StickerFavoriteItem(FavoriteItem)
- **Views**
  - parseFavoriteStickerItems(favoriteStickerItems)
  - parseCartStickerItems(cartStickerItems)
  - FavoritesPage(APIView)
  - AddFavoriteViewSticker(APIView)
  - RemoveFavoriteSticker(APIView)
  - CartPage(APIView)
  - AddStickerToCart(APIView)
  - RemoveStickerFromCart(APIView)

D Application: app_order

- **Functions**
  - This application handles data related to ordering.
- **Models**
  - DeliveryData()
  - Order(DeliveryData)
  - OrderItem()
  - OrderStickerItem(OrderItem)
- **Views**
  - parseOrderCreationPageData(cartStickerItems)
  - parseGetOrderData(orders)
  - OrderCreationPage(generic.CreateAPIView)
  - GetOrders(APIView)

E Application: app_editor

- **Functions**

  - This application is responsible for editor backend functionality.
- **Models**
  - UserEditor()
  - CanvasMaterial(Material)
  - CanvasMaterialOption(MaterialOption)
  - CanvasStyle(Style)
  - CanvasStyleOption(StyleOption)
  - FinishedCanvas()
  - CanvasCutoutImage()
- **Views**
  - parseEditorPageData()
  - EditorPage(APIView)

6.2.1 API request handling

Django REST framework library is being used for API development within the Django framework.

After installing[24], setting up and importing the library, APIView view with specified HTTP methods such as GET or POST has to be created and linked to the URL dispatcher. All that is needed afterward is to call a specific HTTP method from the frontend on a specific URL.

Additionally, if user authentication is required, JWT tokens will have to be specified in the frontend HTTP method and in the APIView view allowed user permission classes[25] would have to be specified such as AllowAny or IsAuthenticated.

However, implementation of login and register view is utilizing generic CreateAPIView class[26] which already provides POST HTTP method handling using serializers[27], so nothing except permission classes, queryset, and serializer_class has to be provided to make these views functional.

The following list comprises all available API URLs, their HTTP methods, views, and their purpose. Views that these URLs utilize will be specified in the functionality subchapter.

- **URL**: 'api/tags/'

  HTTP method: GET

  view: app_sticker_listing.view.GetFilterOptions()

  purpose: This function is being used when a user browses through the available filter options dropdown. As soon as the user types into the dropdown, new options will be available, either existing tags, existing names, or just the typed text. The user is also informed on how many stickers fall into the scope of that particular tag or a name.

- **URL**: 'api/stickers/'

  HTTP method: GET

  view: app_sticker_listing.StickerList()

  purpose: This function is being called when the user has already selected names, tags, and sizes of stickers he wants to browse through. Stickers filtered by tags and names are returned.

- **URL**: 'api/sticker/<int:sticker_id>/'

  HTTP method: GET

  view: app_sticker_listing.StickerPage()

  purpose: This function returns all the data necessary for the creation of a sticker product page.

- **URL**: 'api/favorites/'

  HTTP method: GET

  view: app_cart_favorites.FavoritesPage()

  purpose: Returns user's favorite items.

- **URL**: 'api/favorites/remove/'

  HTTP method: POST

  view: app_cart_favorites.RemoveFavoriteSticker()

  purpose: Removes favorite sticker from user's favorites.

- **URL**: 'api/favorites/add/'

  HTTP method: POST

  view: app_cart_favorites.AddFavoriteSticker()

  purpose: Adds favorite sticker to user's favorites.

- **URL**: 'api/cart/'

  HTTP method: GET

  view: app_cart_favorites.CartPage()

  purpose: Returns user's cart information.

[24] django-rest-framework.org/#installation
[25] django-rest-framework.org/api-guide/permissions/
[26] django-rest-framework.org/api-guide/generic-views/#createapiview
[27] django-rest-framework.org/api-guide/serializers/

- **URL**: 'api/cart/remove/'
  HTTP method: POST
  view: app_cart_favorites.RemoveStickerFromCart()
  purpose: Removes selected user's sticker from the cart.
- **URL**: 'api/cart/canvas/remove/'
  HTTP method: POST
  view: app_cart_favorites.RemoveCanvasFromCart()
  purpose: Removes the selected user's canvas from the cart.
- **URL**: 'api/cart/add/'
  HTTP method: POST
  view: app_cart_favorites.AddStickerToCart()
  purpose: Adds selected sticker to user's cart.
- **URL**: 'api/cart/add/canvas/'
  HTTP method: POST
  view: app_cart_favorites.AddCanvasToCart()
  purpose: Creates finished canvas and adds it to user's cart.
- **URL**: 'api/order/creation/'
  HTTP method: POST
  view: app_order.OrderCreationPage()
  purpose: Returns data used in the page for creating orders such as final order price.
- **URL**: 'api/order/create/'
  HTTP method: POST
  view: app_order.CreateOrder()
  purpose: Creates order from existing items in the cart.
- **URL**: 'api/order/'
  HTTP method: POST
  view: app_order.GetOrders()
  purpose: Returns all of the user's orders.
- **URL**: 'api/editor/'
  HTTP method: POST
  view: app_editor.EditorPage()
  purpose: Returns data necessary for the editor.
- **URL**: 'api/token/'
  HTTP methods: POST
  view: rest_framework_simplejwt.views.TokenObtainPairView(
  purpose: Default view function from rest framework JWT library which returns refresh and access tokens and required authentication arguments username and password.
- **URL**: 'api/token/refresh/';
  HTTP methods: POST
  view: rest_framework_simplejwt.views.TokenRefreshView()
  purpose: Default view function from rest framework JWT library which returns new access token upon providing valid refresh token.
- **URL**: 'api/token/verify/'
  HTTP methods: POST
  view: rest_framework_simplejwt.views.TokenVerifyView()
  purpose: Default view function from rest framework JWT library which returns validity of provided access token by HTTP status codes.
- **URL**: 'api/token/temp/';
  HTTP methods: POST
  view: web_app_django.views.GetTemporaryUserTokens()
  purpose: This function creates a temporary user in the database and returns refresh and access tokens.
- **URL**: 'api/token/login/';
  HTTP methods: POST
  view: web_app_django.views.LoginView()
  purpose: This function takes in email (can also be a username), password and authenticates user using **django.contrib.auth.autheticate** method.
- **URL**: 'api/token/register'
  HTTP methods: POST
  view: web_app_django.views.RegisterView()
  purpose: This function takes in the username, email, and password, and after validating the uniqueness of username and email, and validating their format, a new User instance is created in the database.

6.3 database

Database was designed and reworked using the Miro[28] tool and implemented through Django's models[29] which outlines the database structure and then commits those changes to the dabase, in our case PostgresSQL database.

Current database structure is divided into 5 sections. General database, gallery database, cart and favorites database, order database and editor database. Following list describes those division, the models that are declared in each of these sections, their datatype, and their functions.

6.3.1 General database

General database mostly outlines definitions for parent[30] models that are used in other databases, excluding SizeLimit and User model.

For models attributes see Figure 45.

- **Model: User()**:
  Represents visitors both logged in as well as a temporary.
- **Model: SizeLimit()**:
  An instance of this model is being used for many stickers that share the same printing size limitations. One reason for making such limitations can be that certain types of stickers have different size constraints on how big of a printing space is available, or another reason would be to make sure that there is enough surface for a vacuum gripper to pick the stickers.
- **Model: Material()**:
  Material is a parent class for other material item-specific materials.
- **Model: Style()**:
  Style is a parent class for other item-specific styles. Style defines price per area, is represented by an icon, has a size limit and other defining variables.
- **Model: Tag()**:
  The tag represents a piece of information that can categorize an item such as a sticker in the gallery by some of its distinct features such as theme, color, and others.

[28]miro.com
[29]docs.djangoproject.com/en/4.0/topics/db/models/
[30]Models that are used by other models inherit its definition

- **Model: MaterialOption()**:
  Each item in a gallery can have multiple material options, and this parent class defines its general variables.
- **Model: StyleOption()**:
  Each item in a gallery can have multiple style options, and this parent class defines its general variables.

6.3.2 Gallery database

The Gallery database includes definitions for stickers, its materials, styles, and the gallery itself.

For models attributes see Figure 46.

- **Model: Gallery()**:
  Gallery is a model representing possible different gallery instances. Such galleries are out of the scope of this project, but examples could be that certain users could create their own sticker shops and would be entirely separated from the current gallery.
- **Model: GalleryItem()**:
  Gallery item is a parent class defining general variables which an item in a gallery would use. Different gallery items besides single stickers could be, for example, sticker packs.
- **Model: Sticker(GalleryItem)**:
  Sticker is a child class of GalleryItem and represents a single sticker that can hold many materials and styles.
- **Model: StickerMaterial(Material)**:
  Sticker-specific material definition.
- **Model: StickerStyle(Style)**:
  Sticker-specific style definition.
- **Model: StickerTag(Tag)**:
  Model for sticker-specific tags.
- **Model: StickerMaterialOption(MaterialOption)**:
  Sticker material option defines a connection between Sticker and a StickerMaterial. A sticker can hold multiple material options.

- **Model: StickerStyleOption(StyleOption)**:

  As a child class of StyleOption, it defines sticker-specific fields which will define the sticker style (design) option. Examples of such options can be glossy or matte after we have chosen the vinyl material option for the sticker.

6.3.3 Cart and favorites database

The cart and favorites database outlines the structure for managing the user's cart and favorite items.

For models attributes see Figure 47.

- **Model: Cart()**:

  Cart is a model representing the user's cart. This model is unique, is automatically created, and serves an organizational purpose.

- **Model: CartItem()**:

  CartItem is a parent class defining general variables which an item in a cart would use.

- **Model: StickerCartItem(CartItem)**:

  StickerCartItem is a child class of CartItem and represents a sticker in a cart.

- **Model: CanvasCartItem(CartItem)**:

  CanvasCartItem is a child class of CartItem and represents a user-created canvas in a cart.

- **Model: Favorites()**:

  Favorites is a model representing the user's favorite items. This model is unique, is automatically created, and serves an organizational purpose.

- **Model: FavoriteItem()**:

  Favorite item is a parent class defining general variables which an item saved to favorites would have.

- **Model: StickerFavoriteItem(FavoriteItem)**:

  StickerFavoriteItem is a child class of FavoriteItem and represents a sticker saved as a favorite.

6.3.4 Order database

The order database outlines the structure for managing orders.

For models attributes see Figure 48.

- **Model: DeliveryData()**:

  Delivery data is a model defining basic order information that the user has to provide.

- **Model: Order(DeliveryData)**:

  Model representing an order made by the user.

- **Model: OrderItem()**:

  OrderItem is a parent class defining general variables an item in order would have.

- **Model: OrderStickerItem(OrderItem)**:

  Child model of OrderItem and represents a sticker in a user's order.

- **Model: OrderCanvasItem(OrderItem)**:

  Child model of OrderItem and represents a canvas in a user's order.

6.3.5 Editor database

This database handles data necessary for the editor to function.

For models attributes see Figure 49.

- **Model: CanvasMaterial(Material)**:

  Defines canvas-specific material.

- **Model: CanvasStyle(Style)**:

  Defines canvas-specific style.

- **Model: CanvasMaterialOption(MaterialOption)**:

  Creates a canvas material option for editor users to choose from.

- **Model: CanvasStyleOption(StyleOption)**:

  Creates a canvas style option for editor users to choose from and links it to a specific CanvasMaterialOption.

- **Model: UserEditor()**:

  UserEditor is a model representing the user's editor. This model is unique, is automatically created, and serves an organizational purpose.

- **Model: CanvasCutoutImage()**:

  Represents a cutout option that a user can choose from when deciding what should be a sticker's final shape.

- **Model: FinishedCanvas()**:

  Represents a canvas that the user uploaded to the server. This canvas is later used when displaying the user's canvas in the user's cart and when creating an order.

6.3.6 Visualization of databases

After designing and implementing the database, an application DataGrip[18] from JetBrains was used for viewing the automatically generated ERD (Entity-relationship model) of the PostgresSQL database and positioning its models according to our designed logical structure. This structure was then exported to an application draw.io[12], where final presentational touches were made.

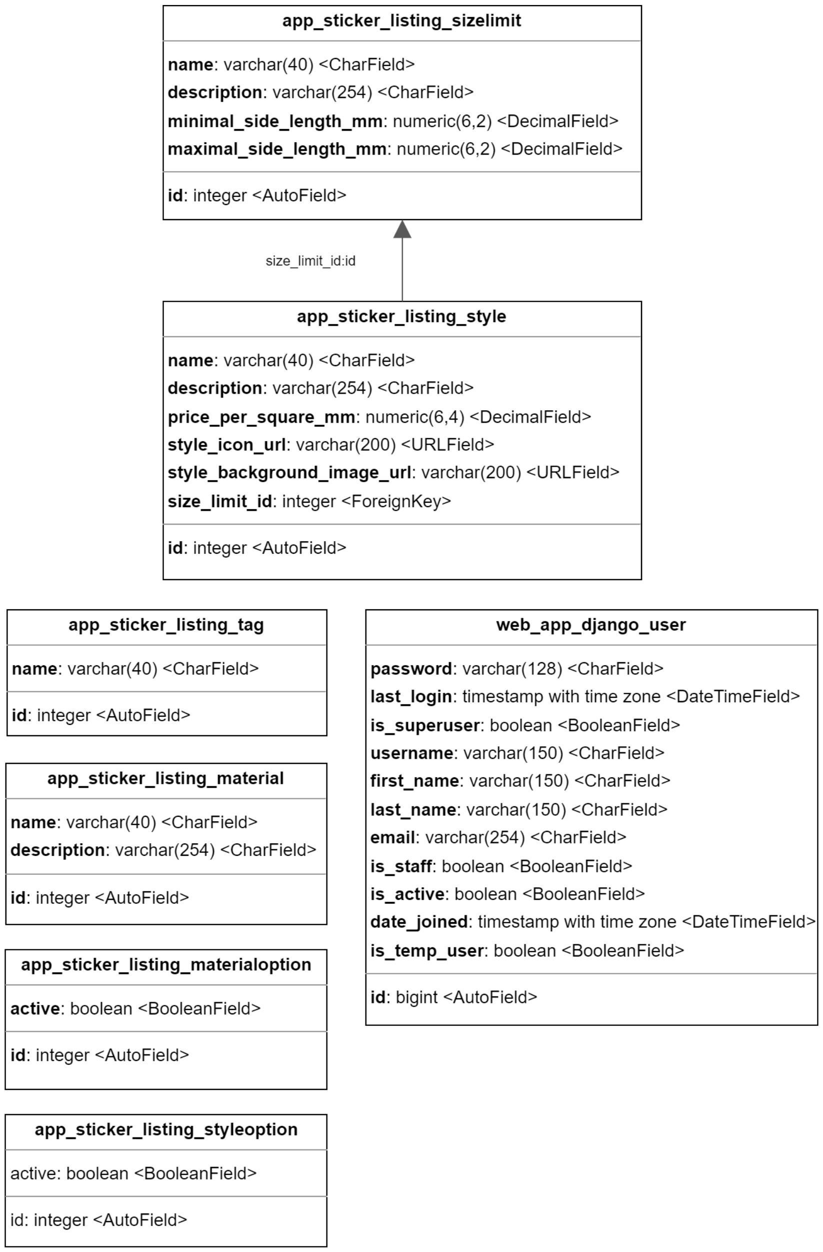

app_sticker_listing_sizelimit
name: varchar(40) <CharField>
description: varchar(254) <CharField>
minimal_side_length_mm: numeric(6,2) <DecimalField>
maximal_side_length_mm: numeric(6,2) <DecimalField>
id: integer <AutoField>
size_limit_id:id
app_sticker_listing_style
name: varchar(40) <CharField>
description: varchar(254) <CharField>
price_per_square_mm: numeric(6,4) <DecimalField>
style_icon_url: varchar(200) <URLField>
style_background_image_url: varchar(200) <URLField>
size_limit_id: integer <ForeignKey>
id: integer <AutoField>
app_sticker_listing_tag
name: varchar(40) <CharField>
id: integer <AutoField>
web_app_django_user
password: varchar(128) <CharField>
last_login: timestamp with time zone <DateTimeField>
is_superuser: boolean <BooleanField>
username: varchar(150) <CharField>
first_name: varchar(150) <CharField>
last_name: varchar(150) <CharField>
email: varchar(254) <CharField>
is_staff: boolean <BooleanField>
is_active: boolean <BooleanField>
date_joined: timestamp with time zone <DateTimeField>
is_temp_user: boolean <BooleanField>
id: bigint <AutoField>
app_sticker_listing_material
name: varchar(40) <CharField>
description: varchar(254) <CharField>
id: integer <AutoField>
app_sticker_listing_materialoption
active: boolean <BooleanField>
id: integer <AutoField>
app_sticker_listing_styleoption
active: boolean <BooleanField>
id: integer <AutoField>

Figure 45: General database structure

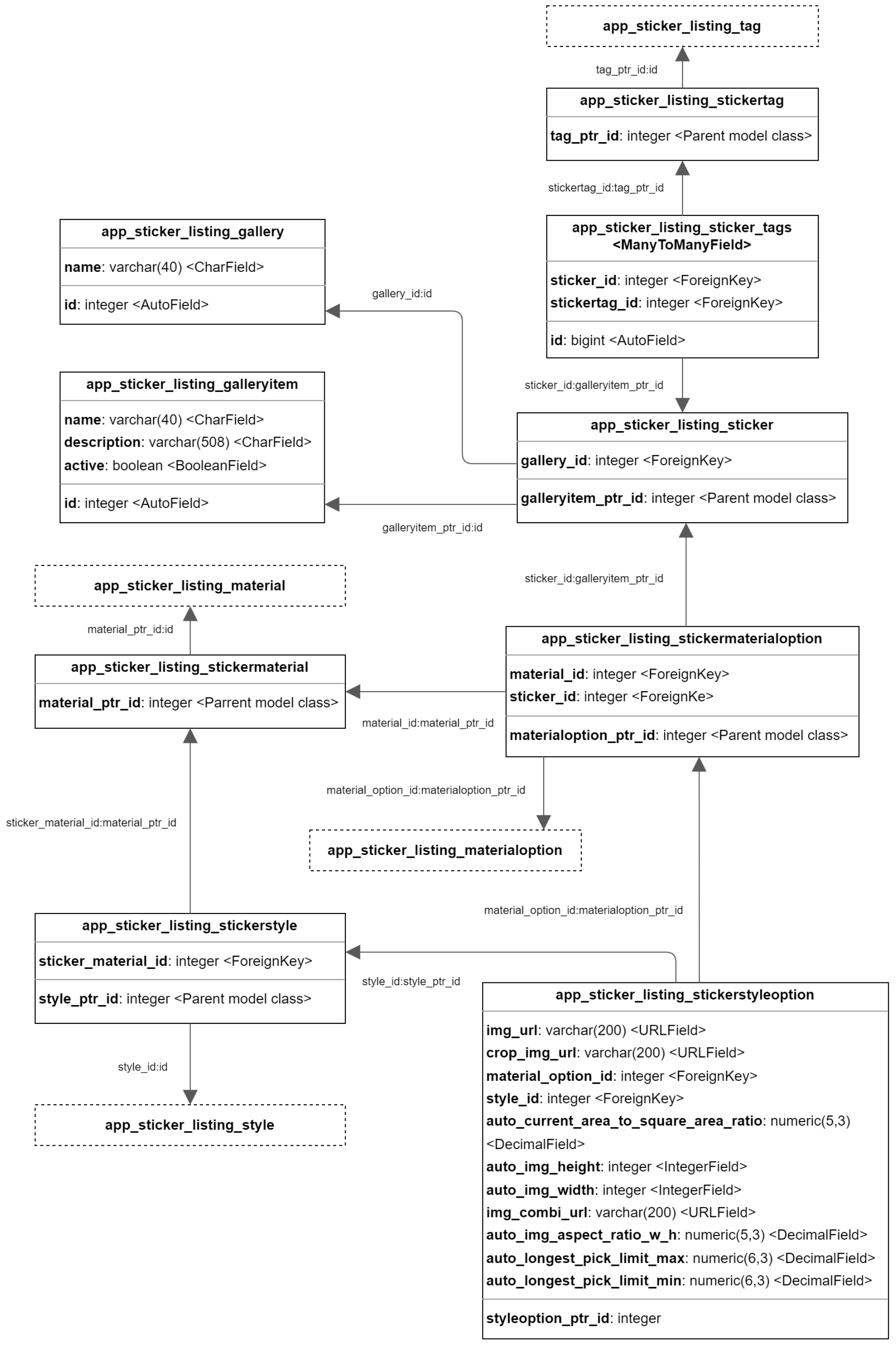

app_sticker_listing_tag
tag_ptr_id:id
app_sticker_listing_stickertag
tag_ptr_id: integer <Parent model class>
stickertag_id:tag_ptr_id
app_sticker_listing_sticker_tags <ManyToManyField>
sticker_id: integer <ForeignKey>
stickertag_id: integer <ForeignKey>
id: bigint <AutoField>
app_sticker_listing_gallery
name: varchar(40) <CharField>
id: integer <AutoField>
gallery_id:id
sticker_id:galleryitem_ptr_id
app_sticker_listing_galleryitem
name: varchar(40) <CharField>
description: varchar(508) <CharField>
active: boolean <BooleanField>
id: integer <AutoField>
app_sticker_listing_sticker
gallery_id: integer <ForeignKey>
galleryitem_ptr_id: integer <Parent model class>
galleryitem_ptr_id:id
app_sticker_listing_material
sticker_id:galleryitem_ptr_id
material_ptr_id:id
app_sticker_listing_stickermaterial
material_ptr_id: integer <Parrent model class>
app_sticker_listing_stickermaterialoption
material_id: integer <ForeignKey>
sticker_id: integer <ForeignKe>
materialoption_ptr_id: integer <Parent model class>
material_id:material_ptr_id
material_option_id:materialoption_ptr_id
sticker_material_id:material_ptr_id
app_sticker_listing_materialoption
material_option_id:materialoption_ptr_id
app_sticker_listing_stickerstyle
sticker_material_id: integer <ForeignKey>
style_ptr_id: integer <Parent model class>
style_id:style_ptr_id
app_sticker_listing_stickerstyleoption
img_url: varchar(200) <URLField>
crop_img_url: varchar(200) <URLField>
material_option_id: integer <ForeignKey>
style_id: integer <ForeignKey>
auto_current_area_to_square_area_ratio: numeric(5,3) <DecimalField>
auto_img_height: integer <IntegerField>
auto_img_width: integer <IntegerField>
img_combi_url: varchar(200) <URLField>
auto_img_aspect_ratio_w_h: numeric(5,3) <DecimalField>
auto_longest_pick_limit_max: numeric(6,3) <DecimalField>
auto_longest_pick_limit_min: numeric(6,3) <DecimalField>
styleoption_ptr_id: integer
style_id:id
app_sticker_listing_style

Figure 46: Gallery database structure

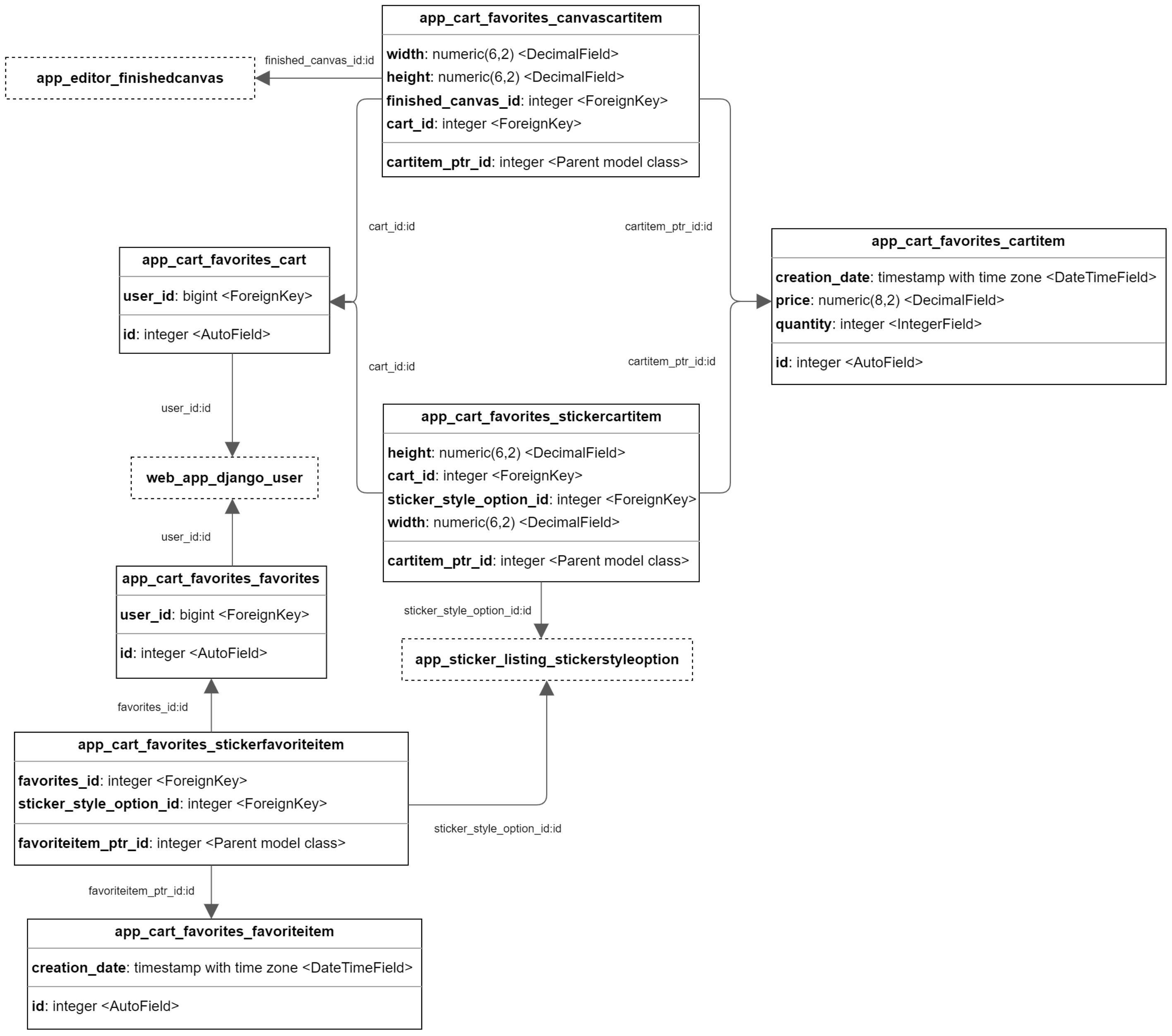

app_cart_favorites_canvascartitem
width: numeric(6,2) <DecimalField>
height: numeric(6,2) <DecimalField>
finished_canvas_id: integer <ForeignKey>
cart_id: integer <ForeignKey>
cartitem_ptr_id: integer <Parent model class>
app_editor_finishedcanvas
finished_canvas_id:id
cart_id:id
cartitem_ptr_id:id
app_cart_favorites_cart
user_id: bigint <ForeignKey>
id: integer <AutoField>
app_cart_favorites_cartitem
creation_date: timestamp with time zone <DateTimeField>
price: numeric(8,2) <DecimalField>
quantity: integer <IntegerField>
id: integer <AutoField>
cart_id:id
cartitem_ptr_id:id
user_id:id
web_app_django_user
app_cart_favorites_stickercartitem
height: numeric(6,2) <DecimalField>
cart_id: integer <ForeignKey>
sticker_style_option_id: integer <ForeignKey>
width: numeric(6,2) <DecimalField>
cartitem_ptr_id: integer <Parent model class>
user_id:id
app_cart_favorites_favorites
user_id: bigint <ForeignKey>
id: integer <AutoField>
sticker_style_option_id:id
app_sticker_listing_stickerstyleoption
favorites_id:id
app_cart_favorites_stickerfavoriteitem
favorites_id: integer <ForeignKey>
sticker_style_option_id: integer <ForeignKey>
favoriteitem_ptr_id: integer <Parent model class>
sticker_style_option_id:id
favoriteitem_ptr_id:id
app_cart_favorites_favoriteitem
creation_date: timestamp with time zone <DateTimeField>
id: integer <AutoField>

Figure 47: Cart and favorites database structure

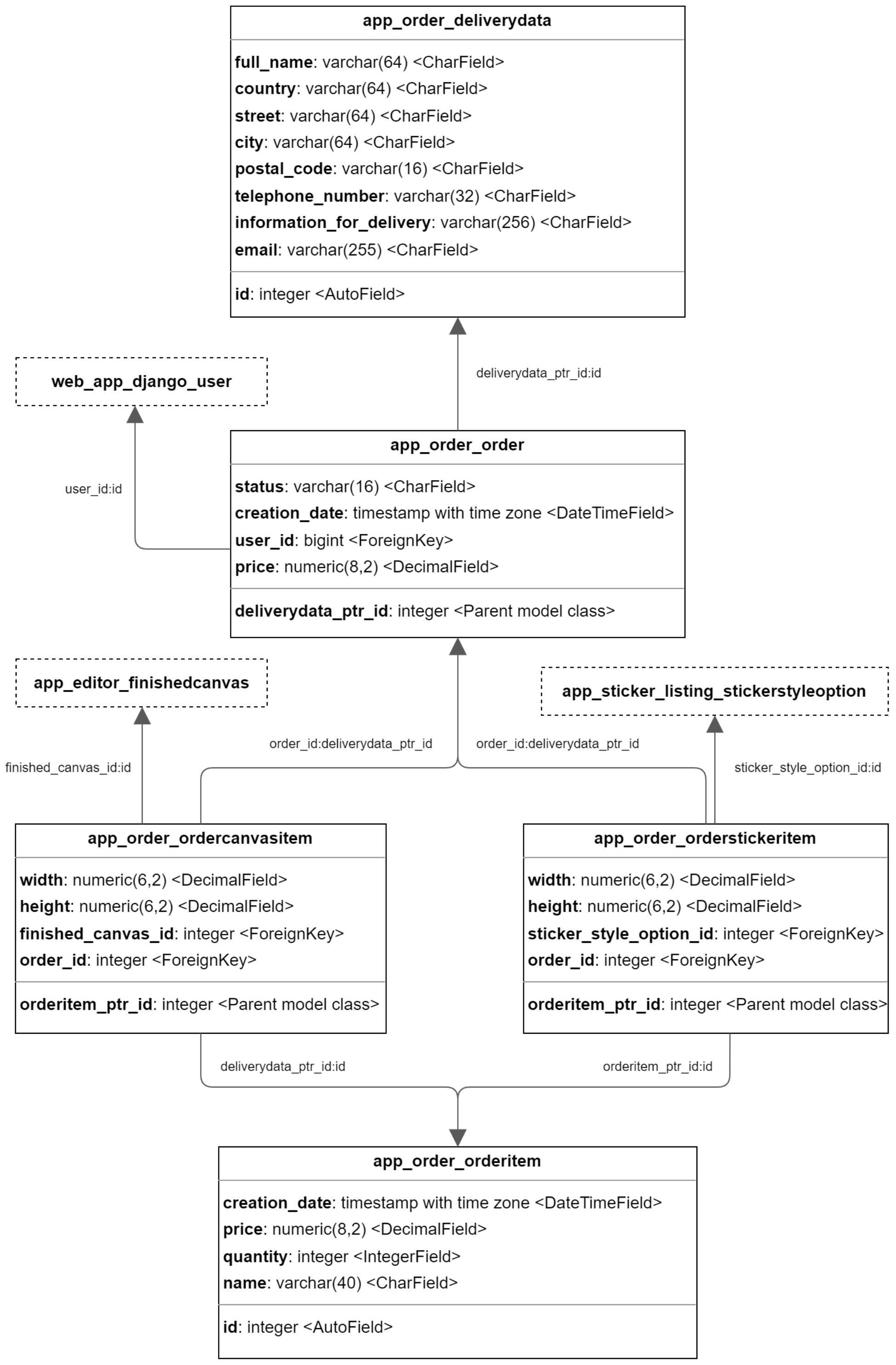

app_order_deliverydata
full_name: varchar(64) <CharField>
country: varchar(64) <CharField>
street: varchar(64) <CharField>
city: varchar(64) <CharField>
postal_code: varchar(16) <CharField>
telephone_number: varchar(32) <CharField>
information_for_delivery: varchar(256) <CharField>
email: varchar(255) <CharField>
id: integer <AutoField>
deliverydata_ptr_id:id
web_app_django_user
user_id:id
app_order_order
status: varchar(16) <CharField>
creation_date: timestamp with time zone <DateTimeField>
user_id: bigint <ForeignKey>
price: numeric(8,2) <DecimalField>
deliverydata_ptr_id: integer <Parent model class>
app_editor_finishedcanvas
app_sticker_listing_stickerstyleoption
order_id:deliverydata_ptr_id
order_id:deliverydata_ptr_id
finished_canvas_id:id
sticker_style_option_id:id
app_order_ordercanvasitem
width: numeric(6,2) <DecimalField>
height: numeric(6,2) <DecimalField>
finished_canvas_id: integer <ForeignKey>
order_id: integer <ForeignKey>
orderitem_ptr_id: integer <Parent model class>
app_order_orderstickeritem
width: numeric(6,2) <DecimalField>
height: numeric(6,2) <DecimalField>
sticker_style_option_id: integer <ForeignKey>
order_id: integer <ForeignKey>
orderitem_ptr_id: integer <Parent model class>
deliverydata_ptr_id:id
orderitem_ptr_id:id
app_order_orderitem
creation_date: timestamp with time zone <DateTimeField>
price: numeric(8,2) <DecimalField>
quantity: integer <IntegerField>
name: varchar(40) <CharField>
id: integer <AutoField>

Figure 48: Order database structure

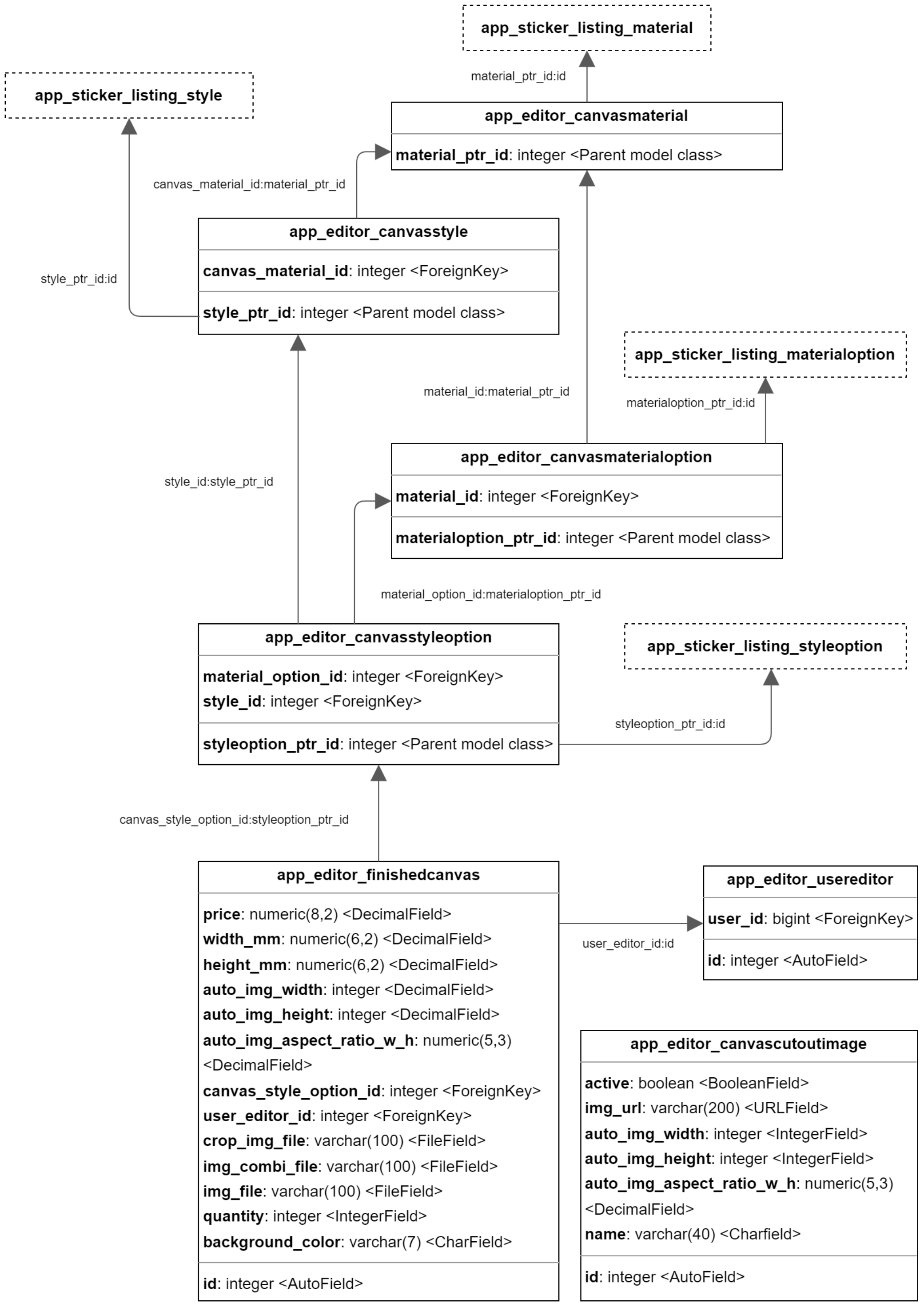

app_sticker_listing_material
material_ptr_id:id
app_sticker_listing_style
app_editor_canvasmaterial
material_ptr_id: integer <Parent model class>
canvas_material_id:material_ptr_id
app_editor_canvasstyle
canvas_material_id: integer <ForeignKey>
style_ptr_id:id
style_ptr_id: integer <Parent model class>
app_sticker_listing_materialoption
material_id:material_ptr_id
materialoption_ptr_id:id
app_editor_canvasmaterialoption
style_id:style_ptr_id
material_id: integer <ForeignKey>
materialoption_ptr_id: integer <Parent model class>
material_option_id:materialoption_ptr_id
app_editor_canvasstyleoption
app_sticker_listing_styleoption
material_option_id: integer <ForeignKey>
style_id: integer <ForeignKey>
styleoption_ptr_id:id
styleoption_ptr_id: integer <Parent model class>
canvas_style_option_id:styleoption_ptr_id
app_editor_finishedcanvas
app_editor_usereditor
price: numeric(8,2) <DecimalField>
user_id: bigint <ForeignKey>
width_mm: numeric(6,2) <DecimalField>
user_editor_id:id
height_mm: numeric(6,2) <DecimalField>
id: integer <AutoField>
auto_img_width: integer <DecimalField>
auto_img_height: integer <DecimalField>
auto_img_aspect_ratio_w_h: numeric(5,3) <DecimalField>
canvas_style_option_id: integer <ForeignKey>
user_editor_id: integer <ForeignKey>
crop_img_file: varchar(100) <FileField>
img_combi_file: varchar(100) <FileField>
img_file: varchar(100) <FileField>
quantity: integer <IntegerField>
background_color: varchar(7) <CharField>
id: integer <AutoField>
app_editor_canvascutoutimage
active: boolean <BooleanField>
img_url: varchar(200) <URLField>
auto_img_width: integer <IntegerField>
auto_img_height: integer <IntegerField>
auto_img_aspect_ratio_w_h: numeric(5,3) <DecimalField>
name: varchar(40) <Charfield>
id: integer <AutoField>

Figure 49: Editor database structure

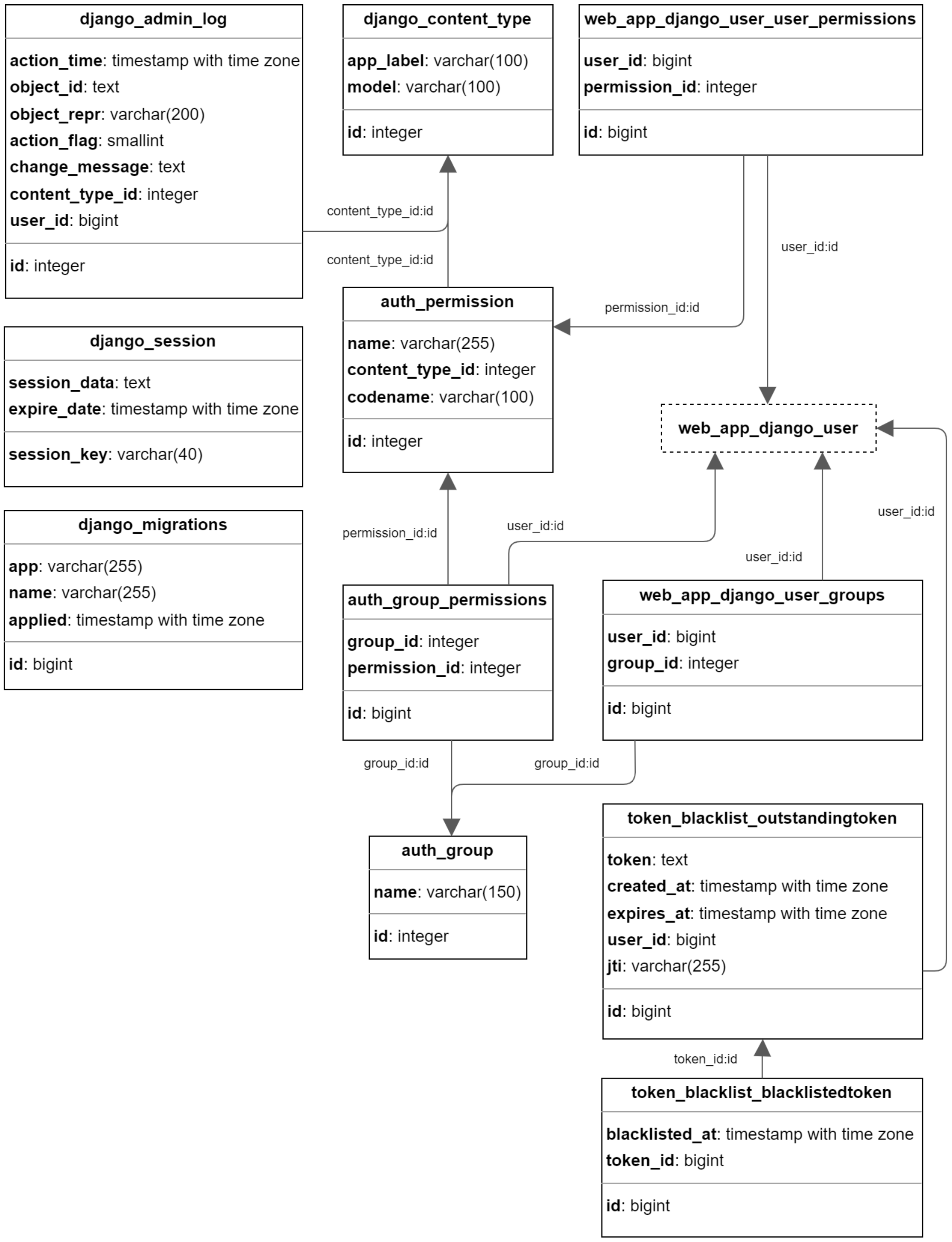


Figure 50: Default Django autogenerated database structure (for completeness)

6.4 functionality

6.4.1 authentication

Authentication is implemented with JWT[31], specifically with JWT library[32] for Django rest framework. This library takes care of the token generation and validation.

In order to pr2ovide user-specific services such as saving items to cart or favorites and without having to implement copies of such saving functions in browser storage (for unauthenticated users) and in the database (for logged-in users), each user has a User model instance in a database. In order for a user to access this instance, the user is provided JWT tokens to access a temporary database user instance, which is created upon visiting the application. The logic for handling this behavior is described in the following diagram.

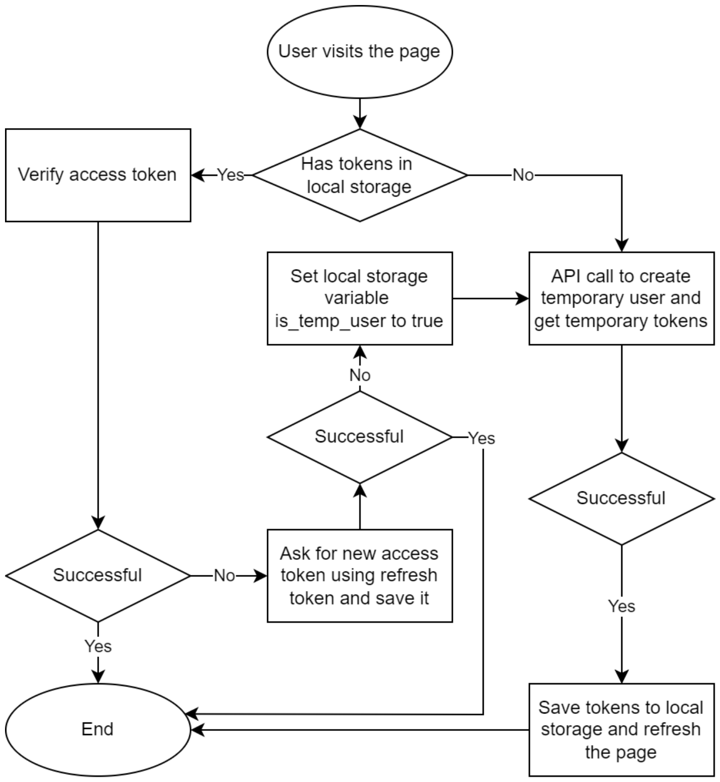

User visits the page
Has tokens in local storage
Yes
No
Verify access token
Set local storage variable is_temp_user to true
API call to create temporary user and get temporary tokens
No
Successful
Yes
Successful
Successful
No
Ask for new access token using refresh token and save it
Yes
Yes
End
Save tokens to local storage and refresh the page

Figure 51: Action diagram for a user who visits the web application page

[31] JSON web tokens
[32] django-rest-framework-simplejwt.readthedocs.io

After a user has obtained valid tokens for the temporary user, he can save items to the cart, interact with canvas, and save items to favorites as a regular logged-in user could.

If, however, the user wants to have an account, have an email, username, and password associated with it. He needs to register.

Logging in and registering is handled by the generic view from Django rest framework library *rest_framework.generics.CreateAPIView* where our own serializers are being used.

In these serializers, we specify what data is required for such API calls, what data is being returned, how such data should be verified, and what types of error text should be returned.

These serializers are defined in a file 'django_root/web_app_django/authentication/serializers.py' and are linked to the specific app that the serializer folder is put in.

6.4.1.1 LoginSerializer

Login serializer firstly tries to authenticate the user with the data is received. If a user is successfully validated, we update the user instance's last login parameter. Else if the user doesn't exist, we return an error message.

After successful authentication, we return his username and tokens.

- **Requiered data**

  Even though email is specified as a required data field, the user can enter his username into this field too. For this functionality to work, the default Django authentication function had to be modified and linked in the Django settings. Function handling authentication is defined in file 'django_root/web_app_django/authentication/UsernameAndE

  - email
  - password

- **Data returned on success**

  On successful login, the user receives a username and tokens for that particular account.

  - username
  - access
  - refresh

6.4.1.2 RegisterSerializer

6.4.1.2 RegisterSerializer Register serializer checks uniquenes of username, email, their minimal lengths, and uses standard internal password validation function[33].

Upon valid validation, the user instance is created, and the user receives valid HTTP OK status code 200.

- **Requiered data**
  - username
  - email
  - password
- **Data returned on success**

  Just HTTP OK status code 200 is returned.

6.4.2 media storage

In order to minimize the server load responsible for returning API responses, all sticker images and other images are stored on a public AWS S3 instance.

When creating a new sticker, we need to specify its location on the S3 instance. These sticker images are arranged in the following fashion.

[33]docs.djangoproject.com/en/2.0/_modules/django/ contrib/auth/password_validation/

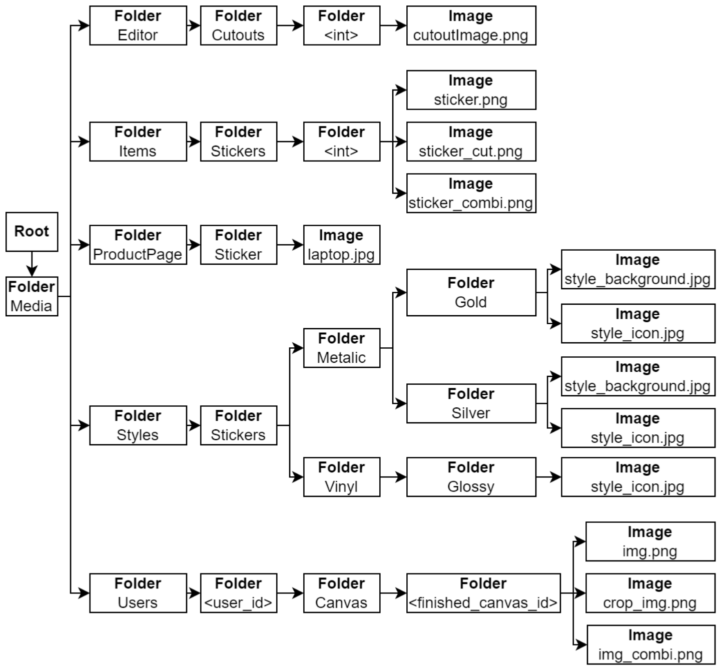


Figure 52: File system structure on media server

6.4.3 creating stickers

In the current state, sticker creation is being handled using the built-in Django admin site[34].

Firstly, we have to specify our models in the admin.py file in each of the Django applications.

After that, if we created Django admin user[35], we can just access the admin site with our credentials by navigating to the admin URL path specified in the URL dispatcher file ('/admin' by default).

A Creating the first sticker

For creating the first sticker, we have to have uploaded sticker images in the AWS S3 instance and also have other model instances available.

Order for actions we have to take in order to create the first sticker.

(a) Create SizeLimit model instance

(b) Create StickerMaterial model instance

[34]docs.djangoproject.com/en/4.0/ref/contrib/admin/

[35]python manage.py createsuperuser

(c) Create StickerStyle model instance
(d) Create few instances of StickerTags
(e) Create Gallery model instance
(f) Create StickerItem model instance and link it to Gallery instance
(g) Create StickerMaterialOption model instance and link it to (StickerItem, StickerMaterial) instance
(h) Create StickerStyleOption model instance and link it to (StickerStyle, StickerMaterialOption)

B Creating another sticker with the same gallery, material and style

For creating another sticker, we don't have to do as many actions as with the first sticker.

(a) Create StickerItem model instance and link it to Gallery instance
(b) Create StickerMaterialOption model instance and link it to (StickerItem, StickerMaterial) instance
(c) Create StickerStyleOption model instance and link it to (StickerStyle, StickerMaterialOption)

6.4.4 filtering system

A user can access the filtering system on the main page.

Users can filter stickers by specifying their name or a tag. Additionally, multiple combinations of these are supported.

Firstly, a user can ask for available tags and names by specifying part of its text and sending HTTP GET method on API URL "api/tags/" . If the text, the user is using to filter the stickers, has at least some threshold length (currently of size ¿= 3), then StickerTags and names are individually gathered, ordered by the count (tags can be part of multiple stickers, and multiple stickers can contain the same text), and the query is limited by a set constant[36].

All of this filtering is being handled by Django query filter[37] and is implemented in a view function GetFilterOptions() in *web_app_django/views.py* views file.

After the user selects the tags and names to filter stickers by, another HTTP GET API call is being made to URL "api/stickers" with tags, names, and selected longest dimension length as parameters. This API call is handled by view function StickerList() in Django *web_app_django* application.

This view function parses the arguments and calls successively functions for filtering all sticker objects by tags and then filtering those filtered stickers by names. After all of this is done, data about individual filtered stickers are sent to the user.

6.4.5 price calculation

Current project architecture utilizes user customization in terms of length and dynamic variable price calculation based on the length of the longest side of the sticker multiplied by sticker style price parameters, in this case, price_per_square_mm variable.

6.4.6 product page

After the user clicks on the listed sticker, he is rerouted to the sticker page. While loading the sticker page, the user is calling an API URL "api/sticker/<int:sticker_id>/" to get the data of the sticker he is visiting. This API call is handled by view function *StickerPage()* of Django app_sticker_listing application.

All that this view function is doing is checking if such sticker instance exists, is active, and, if so, returns correctly parsed data about that sticker. If such a sticker doesn't exist or is not active, the function returns HTTP 404 status code.

6.4.7 favorites

After the user clicks on a sticker in the gallery or on the product page, an API call is being made to the URL "api/favorites/add/" with sticker identity information as well as user authentication. User's favorite items are updated on the client's side with an API call on "api/favorites/" and the button that is used to add that sticker to favorites will remove the sticker from favorite by calling an API URL "api/favorites/remove/" with the same data as with adding.

6.4.8 cart

After the user clicks on the 'add to cart' or 'buy' button, depending on if it is a canvas or a sticker, API calls are being made on URLs "api/cart/add/canvas/" and "api/cart/add/"

[36]For example, a query won't return more than 40 tag instances
[37]docs.djangoproject.com/en/4.0/topics/db/queries/

accordingly. If it's a canvas, image data is being transferred additionally to wanted dimensions, material, style, and user authentication information.

After the user clicks on the remove button on the favorites page, API calls are being made on URLs "api/cart/canvas/remove/" and "api/cart/remove/" depending on if its a canvas or a sticker with their identifying information and user authentication data. After that, the cart item is removed from the database.

6.4.9 ordering

Ordering is being handled with serializers[2]. After submitting the order with the user contact information on API URL "api/order/create/", all the data is being checked with custom validator functions fields[38]. After successful submission, the serializer parses the cart items to order items and creates a new order, as well as removes all the user's cart items from the user's cart.

6.4.10 editor

After the user submits the finished design on API URL "api/cart/add/canvas/", a new instance of model FinishedCanvas is being created. The finished canvas contains all the necessary data to print a sticker, including dimensions, image to be printed, cutout image, and combined image (image + cutout + background). Such images are automatically created after canvas submission in the backend.

After finishing the order, a new instance of the OrderCanvasItem model is created, which parses data from the item's FinishedCanvas.

[38]most of these validators check only the field size but can be easily extended

7 Chapter: Testing

The application was deployed on the Heroku cloud platform (files used in deployment can be found in the source code [20]). The application took up 2 Heroku server instances, one instance was used for the frontend and the other for the backend. Additionally, backend server run the PostgresSQL database Heroku addon [39].

The application was globally accessible and tested on desktop as well as mobile devices. The application is not thoroughly optimized, and because of large uncompressed images and because of frequent API calls, the application is slightly slower on mobile devices.

Additionally, the editor is implemented using [21], and during the testing, it was discovered that this element is being rendered much faster on the Google Chrome browser compared to Firefox as the canvas element is utilizing Google Chrome's hardware acceleration [22].

The editor was implemented for the desktop version of the application and is thus unusable using mobile devices. Other parts of the application work optimally[40] on both desktop and mobile.

As one of the application's goals was to be utilized in the automatic printing of stickers, a Python script that connects to the database and prints the ordered items on paper was implemented and tested. Video of the actual test of the application and automated printing is located in the source code directory [20].

[39]elements.heroku.com/addons/heroku-postgresql
[40]besides the slow speed on mobile devices mentioned earlier

8 Chapter: Running the application

There are two recommended options for running the application. Both of these solutions require setting up AWS S3 media server[6] and PostgresSQL database[4].

The first option is to run the application on Heroku[5] cloud platform. The prepared files needed to run the application on Heroku are located in the application source code directory [20].

The second recommended option is to run the application locally. This option requires installing Django[1], Node.js, and additional libraries for Python. Requirements are described in the application source code directory [20].

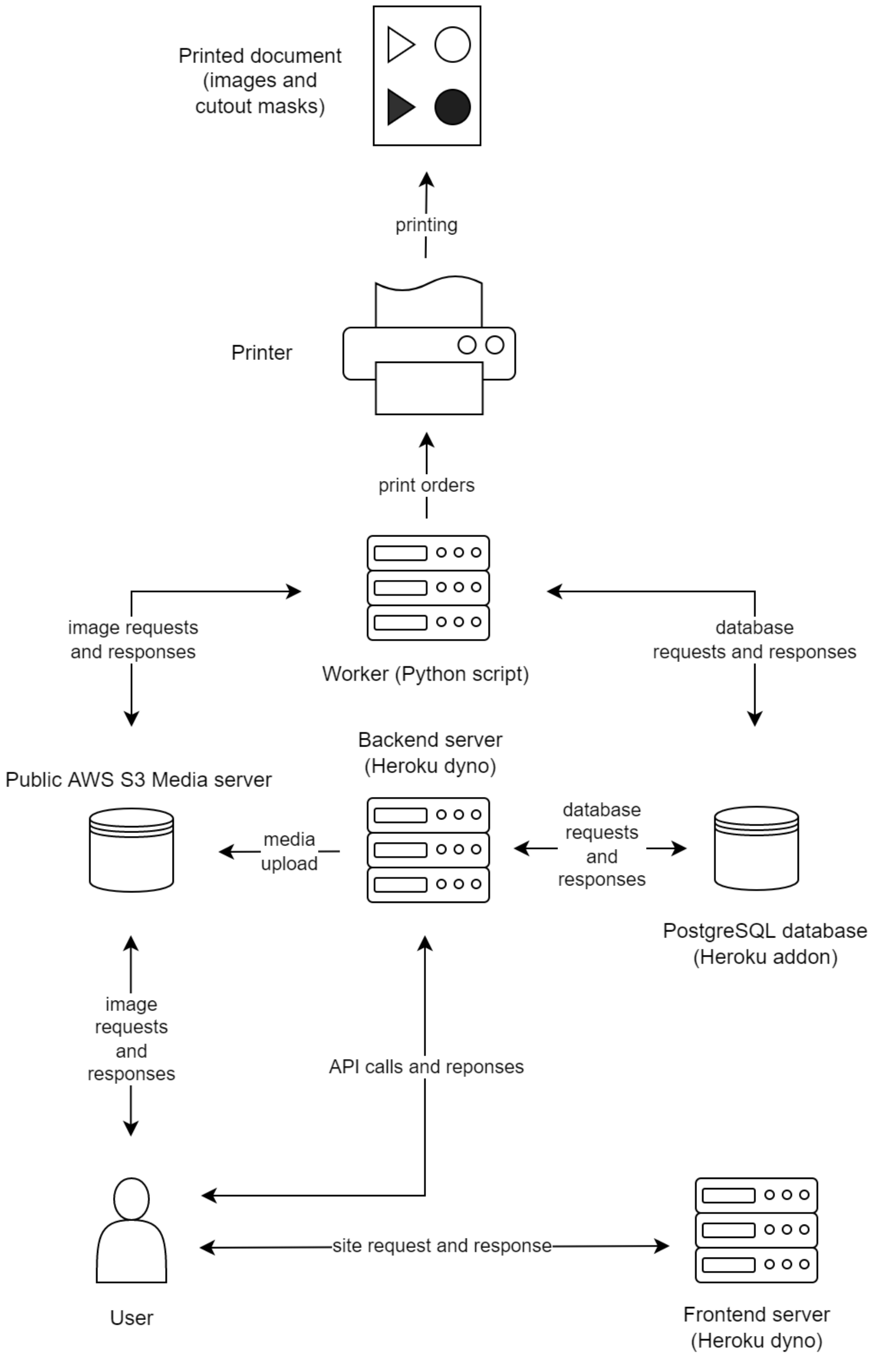

Printed document (images and cutout masks)
printing
Printer
print orders
image requests and responses
Worker (Python script)
database requests and responses
Backend server (Heroku dyno)
Public AWS S3 Media server
media upload
database requests and responses
PostgreSQL database (Heroku addon)
image requests and responses
API calls and reponses
site request and response
User
Frontend server (Heroku dyno)

Figure 53: Architecture of the deployed application

9 Chapter: Future works

Future works that would utilize this application could make use of the entirety of the application and continue developing automated printing on an industrial-grade printer and cutting plotter. Other similar products besides stickers that could use most of the application might include custom laser engraving, woodcutting, metal cutting, glass cutting, and other materials that would require a 2D editor.

Moreover, this work can be partially used in developing automated solutions for different products other than stickers, such products may include custom furniture, sportswear, clothing, and many other solutions that could tap into many different industries and sectors.

Results of this work will be advertised to all contacted companies producing stickers, and thus this work can be used by them when developing their custom solution.

10 Chapter: Conclusion

Part of this work was conducting research of existing solutions and companies providing sticker printing services. The total number of surveyed companies was 25. Most of the researched companies were local, didn't offer an editor for uploading and editing designs, did not offer a gallery with existing designs, and their prices indicated that they are mainly focused on the businesses, not individuals with small volume custom orders as the prices for single small sticker were in the orders of tens of euro.

Some of the researched companies and others were contacted. Totally 19 companies were contacted by email, and 2 of these companies responded. Both of these companies were local. One company was very interested in such a solution as the communication with customers was one of the most expensive parts of the whole ordering process and took on average 45 minutes per customer as described in Section 2.3. The second company was a little skeptical from experience when it comes to offering services for an average person as they often don't provide the best quality files for printing.

After taking existing solutions into consideration, the design of the frontend and backend parts of the application have been developed.

What followed was the implementation of the design using Python (Django framework [1]) for the backend, PostgresSQL [4] database, and JavaScript (React framework [3]) for the frontend. Application has been developed to be suited for both mobile and desktop devices, with the exception of the editor, which is only usable on desktops. Source code can be found in the bibliography[20].

Functional application was then launched on Heroku cloud platform [5] a will be publicly available for at least one year on `sticker-application-frontend.herokuapp.com` and `sticker-application-backend.herokuapp.com`[41]. Additionally, AWS S3 [6] instance has been used as a media server hosting sticker designs.

The application was successfully tested and successfully performed functions described in Section 3. However, it was discovered that the editor, which was implemented using canvas HTML element [21], was much faster than Chrome browser compared to Firefox because of the browser implementation differences [22].

Finally, a Python script that fetches paid orders from the database and parses the sticker images and cutout images, and automatically prints them using a locally connected printer was implemented and tested. Video of the test is available in the directory with the source code [20].

The next step, which is not part of this thesis, is to share the solution and the source code with the local and foreign companies with the hope that they will be heavily inspired to implement their own and try to offer their customers a better service and more customization as well as save the production costs.

References

[1] Official documentation for Python Django `https://docs.djangoproject.com`

[2] Django REST framework serializers `https://www.django-rest-framework.org/api-guide/serializers/`

[3] Official React website `https://reactjs.org`

[4] Official PostgresSQL database website `https://www.postgresql.org/`

[5] Heroku cloud hosting website `https://www.heroku.com/`

[6] Amazon AWS S3 storage website `https://aws.amazon.com/s3/`

[41]see [20] for possible updates related to accessibility

[7] Repository for Node.js package that combines images
http://github.com/lukechilds/merge-images

[8] Redbubble shop website https://www.redbubble.com/

[9] Diginate shop website https://diginate.com/

[10] Stickeryou shop website https://www.stickeryou.com/

[11] Tool used in planning phase of the project
https://miro.com

[12] Tool used in finalizing diagrams for database, architecture, and page visit diagram
https://drawio-app.com/

[13] Design tool Figma that was used in the planning phase of the development https://www.figma.com/

[14] Library of React styled components that were used in frontend implmentation https://mui.com/

[15] Repository of Axios Node.js package for handling HTTP requests https://github.com/axios/axios

[16] Website of Router Node.js package for URL handling in React https://reactrouter.com/

[17] Visual editing application used when editing stickers
https://www.adobe.com/products/photoshop.html

[18] Database tool used when creating database presentational diagrams
https://www.jetbrains.com/datagrip/

[19] Feldroy D., Feldroy A., 24.3.6 Things That Should Be Tested, Two scoops of django 3.x, (ed. 07 2021, pp. 300-301)

[20] Application source code hosted on university GitLab
https://gitlab.fel.cvut.cz/basarpet/bachelor-thesis-sticker-web-application

[21] Editor is implemented using canvas html element
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

[22] Canvas element in Chrome utilizes hardware acceleration
https://developer.chrome.com/blog/taking-advantage-of-gpu-acceleration-in-the-2d-canvas/