Web Application for Automatic Newsletter Distribution

Bc. Jakub Vitásek

Master's thesis
2021

Tomas Bata University in Zlín
Faculty of Applied Informatics

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2020/2021

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jakub Vitásek**
Osobní číslo:	**A18281**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Kombinovaná**
Téma práce:	**Webová aplikace pro automatické rozesílky newsletteru**
Téma práce anglicky:	**An Automatic Newsletter Distribution Web Application**

Zásady pro vypracování

1. Proveďte průzkum trhu a analyzujte výhody a nevýhody existujících řešení.
2. Sumarizujte best practices rozhraní a funkcionality mailing systémů.
3. Implementujte vybrané moduly usnadňující rozesílání newsletterů.
4. Publikujte výsledný kód na vhodném otevřeném repositáři.
5. Věnujte pozornost zabezpečení aplikace.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:

1. GAMMA, Erich, John VLISSIDES a Richard HELM. Design Patterns: Elements of Reusable Object-Oriented Software. 1994. United States: Addison-Wesley, 1994. ISBN 9780321700698.
2. ROMER, Michael. PHP Persistence: Concepts, Techniques and Practical Solutions with Doctrine. 2016. New York: Apress, 2016. ISBN 9781484225585.
3. LABRECQUE, Tammi. Newsletter Ninja: How to Become an Author Mailing List Expert. 2018. ?: Larks and Katydids, 2018. ISBN 099821275X.
4. SNYDER, Chris, Thomas MYER a Michael SOUTHWELL. Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses. 2010. ?: Springer-Verlag, 2010. ISBN 1430233184.
5. MCLAUGHLIN, Molly a Gadjo SEVILLA. The Best Email Marketing Software for 2020. PC Mag [online]. 2020, 2020(?), 1 [cit. 2020-11-26]. Dostupné z: https://www.pcmag.com/picks/the-best-email-marketing-software

Vedoucí diplomové práce:	**doc. Ing. Jiří Vojtěšek, Ph.D.** Ústav řízení procesů
Datum zadání diplomové práce:	**15. ledna 2021**
Termín odevzdání diplomové práce:	**17. května 2021**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D. v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

ABSTRAKT

Cílem této práce je analýza existujících řešení ve sféře komerčních platforem pro automatickou rozesílku newsletteru a představení výsledného open-source systému, vyvinutého na základě analýzy trhu. Výsledná webová aplikace je implementována v PHP 8 za pomoci Nette Frameworku, Doctrine ORM a Vue.js, a byla publikována na platformách Github a Packagist. Aplikace nabízí unikátní editor newsletteru a mnoho dalších funkcionalit, které jsou mezi komerčními platformami dnes již standardní. Při vývoji byl kladen důraz na bezpečnost aplikace a kvalitu kódu. Výsledkem této práce je systém, který dalším vývojářům umožní vytvořit si jeho instanci pro své klienty a případně si jej dále upravovat.

Klíčová slova: Webová aplikace, Nette Framework, Doctrine ORM, Vue.js, mailing systém, automatická rozesílka newsletteru, open-source

ABSTRACT

This thesis aims to give a comprehensive account of the current commercial mailing platforms and present the open-source system implemented based on the market analysis. The resulting web application was built in PHP 8 using Nette Framework, Doctrine ORM and Vue.js, and was published on platforms Github and Packagist. The system offers a unique mailing editor and many functionalities considered standard in commercial platforms, while focusing on security and code quality. As a result of this thesis, developers can use this system to create a self-hosted mailing platform for their clients.

Keywords: Web Application, Nette Framework, Doctrine ORM, Vue.js, Mailing System, Newsletter System, Open-Source

In loving memory, this thesis is dedicated to Miroslav Smékal, an exceptional father, grandfather and husband.

TABLE OF CONTENTS

INTRODUCTION

Marketing e-mails are the driving force of every e-commerce project. When done right, mailings can bring potential clients to a website with a high conversion rate at a fairly low expense. The market is well aware of that fact and takes advantage of the opportunity. As a result, there are basically no platforms offering a free plan and all of them have severe limitations for the non-paying customer.

The goal of this thesis is to design and create an open-source platform that will fill this gap. The resulting system has to provide the standard features like an intuitive dynamic editor, conversion tracking, user segmentation and sendout planning. On the technical side, the system has to be written utilizing the latest technology – so that other programmers can extend it easily to meet their specific needs. There is also the rising requirement of proper security, since mailing systems often deal with sensitive personal data.

In the theoretical part, this thesis describes existing commercial platforms and their respective pros and cons. Furthermore, it goes over existing open-source platforms and the reason why these are not sufficient for today's regular user. The next chapter deals with best practices of the aforementioned platforms, both for user interface (UI) and the system's functionality. Finally, the thesis explores all the necessary requirements for both application and data security.

The first chapter of the practical part shows results of modeling the system entities using the Object-relational mapper (ORM) Doctrine 2 as well as via Unified Modeling Language (UML) diagrams. The next chapter carefully describes important technologies essential for the implementation process and the resulting system itself. Lastly, the thesis mentions both interface and unit tests created to allow for seamless CI/CD processes.

I. THEORETICAL PART

1 MARKET RESEARCH

There are many mailing platforms currently available on the market, ranging from minimalist dashboards to sophisticated systems. The core features this thesis will be looking at are these:

- Starting Price: this is the price of the lowest-level premium plan, which unlocks various extra features and augments the volume of the maximum number of recipients and groups (in some cases also called audiences).
- Free Plan: if a platform offers a free plan, it means a client can use the system without providing his credit card information, but with some limitations, e.g. a certain maximum total number of recipients (also called subscribers).
- Responsive Test: this means that the user is able to preview his mailings in a mobile, tablet or desktop view (more in Chapter 2.1.9).
- Automation: automation provides a way for the user to schedule e-mails based on the recipient's activity – e.g. when a specific link is clicked, the recipient receives a follow-up e-mail.
- Segmentation: this means the user can categorize recipients into segments based on conditional statements and use these segments to address mailings (more in Chapter 2.2.5).
- Planning: when a system provides the planning feature, the user can schedule the sendout of his mailing to execute at a given time (more in Chapter 2.2.6).

The comparison of these features across the mainstream platforms is shown in the Table 1.1.

Table 1.1 Existing Platforms Feature Comparison

	Mailchimp	SendInBlue	CC	Ecomail
Starting Price	$29	$25	$20	$4.5
Free Plan	✓	✓	✗	✓
Responsive Test	✓	✓	✓	✓
Automation	✓	✓	✓	✓
Segmentation	✓	✓	✓	✓
Planning	✓	✓	✓	✓

It is important to note that all the platforms mentioned provide all the core features that clients expect. This is the basis for best practices analysis in Chapter 2. Outside of similarities in best practices, there are some difference in the price of the offered plans and the possibility of creating a free account with a baseline for the maximum number of subscribers. Other distinctions are of a more cosmetic variety, like a more original interface or an intuitive mailing planner.

1.1 Global Commercial Platforms

There are many popular world-renowned platforms that come to mind when discussing mailing systems (also known as newsletter systems). Each of them has its own specialities and focuses on different types of users. An interesting graph showing trends in Google searches for mainstream mailing platforms is shown in Figure 2.4.

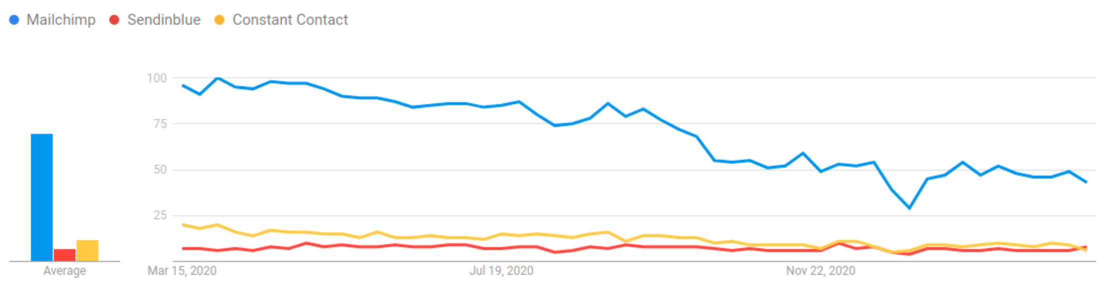


Figure 1.1 Google Trends Statistics for Selected Platforms

The client distribution of these platforms can be seen in Table 1.2, with Mailchimp holding the majority of the mailing system market share. Each of these platforms is unique with its own place on the market, targeting differently sized businesses and different client types.

Table 1.2 Market Share of Global Mailing Platforms [1]

Platform	Market Share
Mailchimp	69.58%
Constant Contact	5.99%
SendInBlue	0.77%

1.1.1 Mailchimp

Dubbed as the "gold standard" [2] among mailing systems, Mailchimp[1)] is the most used global commercial platform to date. Founded in 2001, Mailchimp was designed

1)Mailchimp `https://mailchimp.com/`

as an alternative to the oversized, expensive email software of the early 2000s. It gave small business owners who lacked the high-end tools and resources of their larger competitors access to technology that empowered them and helped them grow. [3] What really sets it apart from others is a plethora of integration options, allowing users to connect their account to applications and tools they are already using. The main advantages and disadvantages of Mailchimp are summarized in the Table 1.3.

Table 1.3 Pros and Cons of Mailchimp

Pros	Cons
Intuitive UI	Sendout schedule only in paid version
In-depth statistics	High cost of premium plans
Reliable mailing editor	

When it comes to the mailing editor, this thesis takes a lot of inspiration from Mailchimp. Mailchimp's editor is trendsetting: it is simple, easy to use, reliable and effective. It provides the right components and allows users to customize necessary attributes.

1.1.2 SendInBlue

This mailing platform is a relative newcomer (being founded in 2012), with the least market share of all the systems mentioned in this thesis – but that does not mean it is in any way substandard. SendInBlue[2)] currently supports more than 180 000 active clients across 160 countries [4] and primarily focuses on SMBs (Small and Midsize Businesses). Table 1.4 shows the main pros and cons of SendInBlue.

Table 1.4 Pros and Cons of SendInBlue

Pros	Cons
Wide automation range	Time-consuming initial setup
A/B testing capabilities	Mailing builder needs work

1.1.3 Constant Contact

Constant Contact[3)] flaunts a minimalist and intuitive interface, providing the majority of standard features like responsive preview and sendout planning (which works in the free plan, in contrast to Mailchimp). As a bonus, Constant Contact offers extra features which allow users to create landing pages, connect e-mail accounts, chat with website visitors in real time or a complete CRM (Customer Relationship Management).

2)SendInBlue `https://www.sendinblue.com/`

3)Constant Contact `https://www.constantcontact.com/`

Unfortunately, since the year 2019, users have been voicing complaints[4)] that there is no way to cancel their account through the dashboard (only accessible when a credit card information is provided). Only by calling Constant Contact's support center and being on hold for many minutes, people report their account has finally been suspended. This seems like a very dishonest tactic devised to retain a maximum of clients by Constant Contact. Pros and cons of the system can be found in the Table 1.5.

Table 1.5 Pros and Cons of Constant Contact

Pros	Cons
Intuitive UI	Limited automation capabilities
Social media integration	Only basic customization of templates
	Dishonest behavior of Constant Contact

1.2 Czech Commercial Platforms

For business owners with local branches only, it may be beneficial to use a local mailing platform. Each country has its own specifics, including different e-shop box solutions, language abnormalities and even own social networks. In the Czech market, it is a good idea to use a Czech mailing system for these reasons:

- They offer integrations for local e-shop solutions, like Shopsys[5)], oXyShop[6)] and Shoptet[7)].
- They are optimized for the local market, enhancing deliverability of mailings.
- They support special traits of the Czech language, like declension.
- They support local name days.
- They can detect the recipient's gender based on local names.
- They seamlessly integrate into local CSEs (Comparison Shopping Engines) like Heureka[8)] and Zbozi[9)].
- They provide documentation and the system's interface in Czech language, which could be a big advantage for users who do not speaking English.

[4)] User complaints against Constant Contacts `https://community.constantcontact.com/t5/Account-Settings-Billing/Cancel-online/idi-p/330769`

[5)] Shopsys `https://www.shopsys.cz/`

[6)] oXyShop `https://www.oxyshop.cz/`

[7)] Shoptet `https://www.shoptet.cz/`

[8)] Heureka `https://www.heureka.cz/`

[9)] Zbozi `https://www.zbozi.cz/`

1.2.1 Ecomail

This local platform largely focuses on deliverability to free Czech email services like Seznam.cz and Volny.cz, and it centers its features around e-commerce businesses. For a relatively new system (founded in 2015 [5]), Ecomail[10)] surprisingly has all the important functionality seen in global platforms such as Mailchimp paired with a sleek, intuitive and reliable user interface. In its free version, a user can have a maximum of 200 recipients with a ceiling of 200 sent e-mails per month. For unlimited use of the platform (including no limitation of the amount of sent e-mails), the user has to upgrade to a paid plan for approximately $4.5 – the price comparison was shown in Table 1.1.

An interesting feature that Ecomail is currently developing is machine learning[11)], which could primarily help big e-commerce accounts to recommend products. With plenty of order data and with the help of web-tracking, Ecomail would be able to train the AI (artificial intelligence) to effectively generate product purchase recommendations for each recipient. This could give Ecomail the edge to be the most advanced mailing system in Czechia.

1.3 Open-source platforms

Since the resulting PHP[12)] (Hypertext Preprocessor) application of the practical part of this thesis (called NWSLTR) is open-source, it makes sense to explore open-source self-hosted platforms (platforms which allow developers to set up their own instance of the system). During this research, only few packages met the criteria of a self-hosted mailing system, with only one being written in PHP.

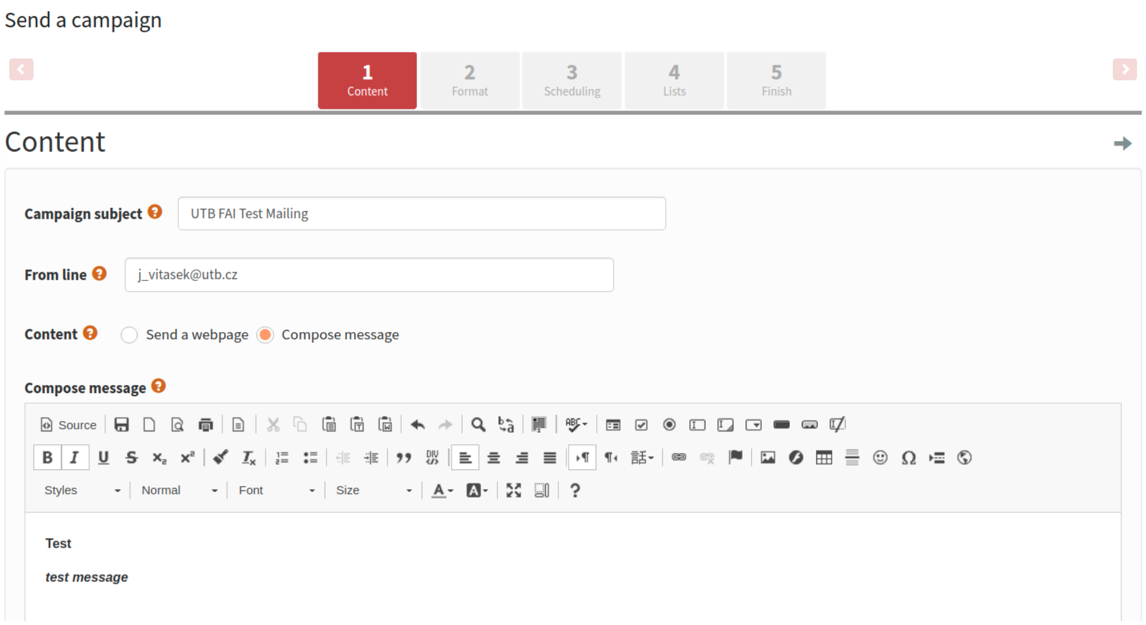
1.3.1 phpList

The application closest to the attributes of the practical part of this thesis is phpList[13)]. At the time of writing this, phpList was in version 3.6.0, released on January 11, 2021. There is a small community around the package, maintaining its core. The package is decoupled, separating the actual core application from the front facade. Unfortunately, the whole admin module is largely written without OOP (Object-oriented Program-

[10)]Ecomail `https://ecomail.cz/`
[11)]Ecomail Features `https://www.ecomail.cz/features`
[12)]PHP `https://www.php.net/`
[13)]phpList `https://www.phplist.org/`

ming) in pure PHP, mixing HTML with back-end code and using old, ineffective and disorganized structures. The core application uses Symfony[14)] with Doctrine[15)] and is a little closer to a modern PHP solution.

UI is where phpList has its biggest problems, from an unattractive and outdated user environment to the mailing editor. Instead of the standard dynamic drag'n'drop user-friendly editor, there is only a single WYSIWYG-enabled (What You See Is What You Get) textarea where a user is supposed to build the mailing template, as visible in Figure 1.2. The whole front-end of this package really lags behind what is expected in the year 2021, making it usable only for developers who code the HTML template themselves and just need a means of sending it.



Campaign subject: UTB FAI Test Mailing

From line: j_vitasek@utb.cz

Content: ( ) Send a webpage [✓ Compose message]

Compose message

Test

test message

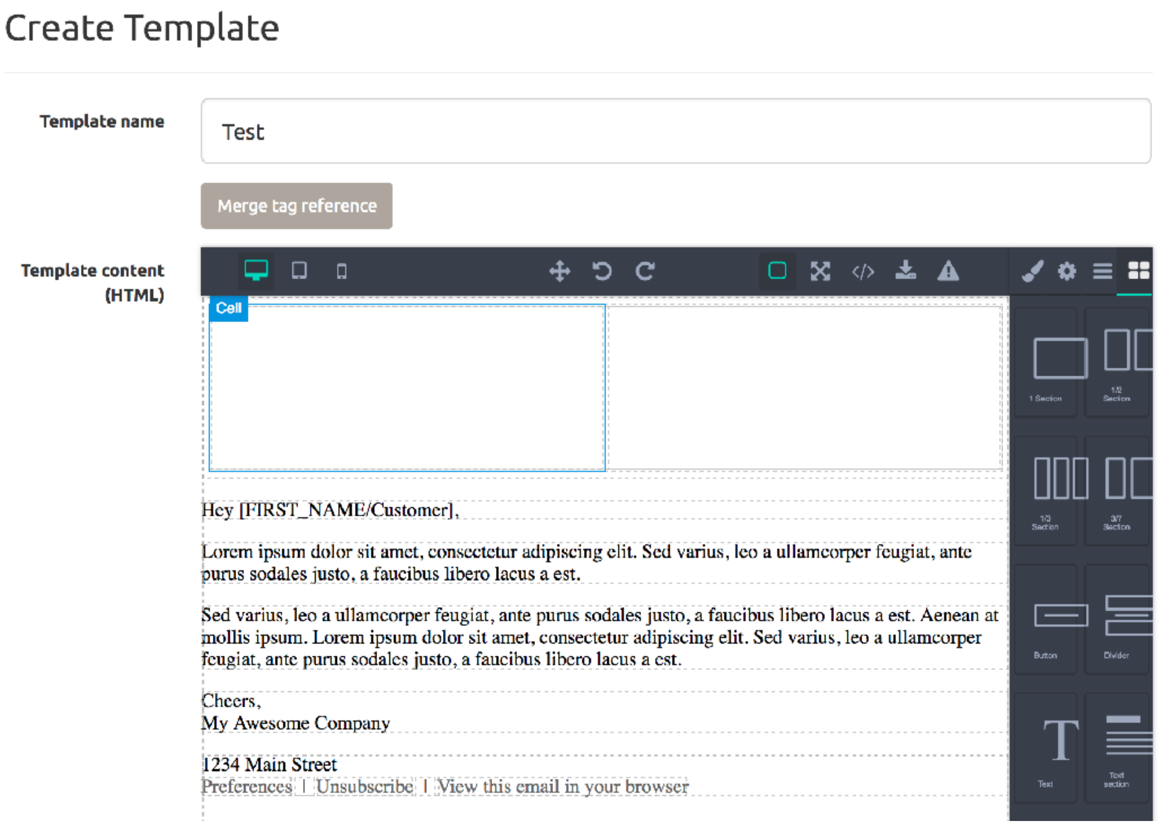
Figure 1.2 Mailing Editor in phpList

1.3.2 Mailtrain

Compared to phpList, Mailtrain[16)] is a bit more modern package, built on Node.js[17)] and MySQL 8[18)] / MariaDB 10[19)]. Even though this thesis looks primarily into PHP platforms, it is fair to say more about Mailtrain. It offers standard functionality, ranging from list management, segmentation and a basic template editor.

14)Symfony `https://symfony.com/`
15)Doctrine `https://www.doctrine-project.org/`
16)Mailtrain `https://mailtrain.org/`
17)Node.js `https://nodejs.org/en/`
18)MySQL 8 `https://www.mysql.com/`
19)MariaDB `https://mariadb.org/`

The problems arise when users try to schedule a specific date for a sendout, which is not possible, with users requesting this feature since the year 2017. Mailtrain's mailing editor uses two open-source packages: GrapeJS[20] and Mosaico[21]. This puts it a little closer to the current standards of its commercial counterparts – and while these packages work fine, the editor design is quite outdated (as displayed in Figure 1.3), generating unattractive e-mail templates.



Test

Hey [FIRST_NAME/Customer],

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed varius, leo a ullamcorper feugiat, ante purus sodales justo, a faucibus libero lacus a est.

Sed varius, leo a ullamcorper feugiat, ante purus sodales justo, a faucibus libero lacus a est. Aenean at mollis ipsum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed varius, leo a ullamcorper feugiat, ante purus sodales justo, a faucibus libero lacus a est.

Cheers,
My Awesome Company

1234 Main Street
Preferences | Unsubscribe | View this email in your browser

Figure 1.3 Mailing Editor in Mailtrain

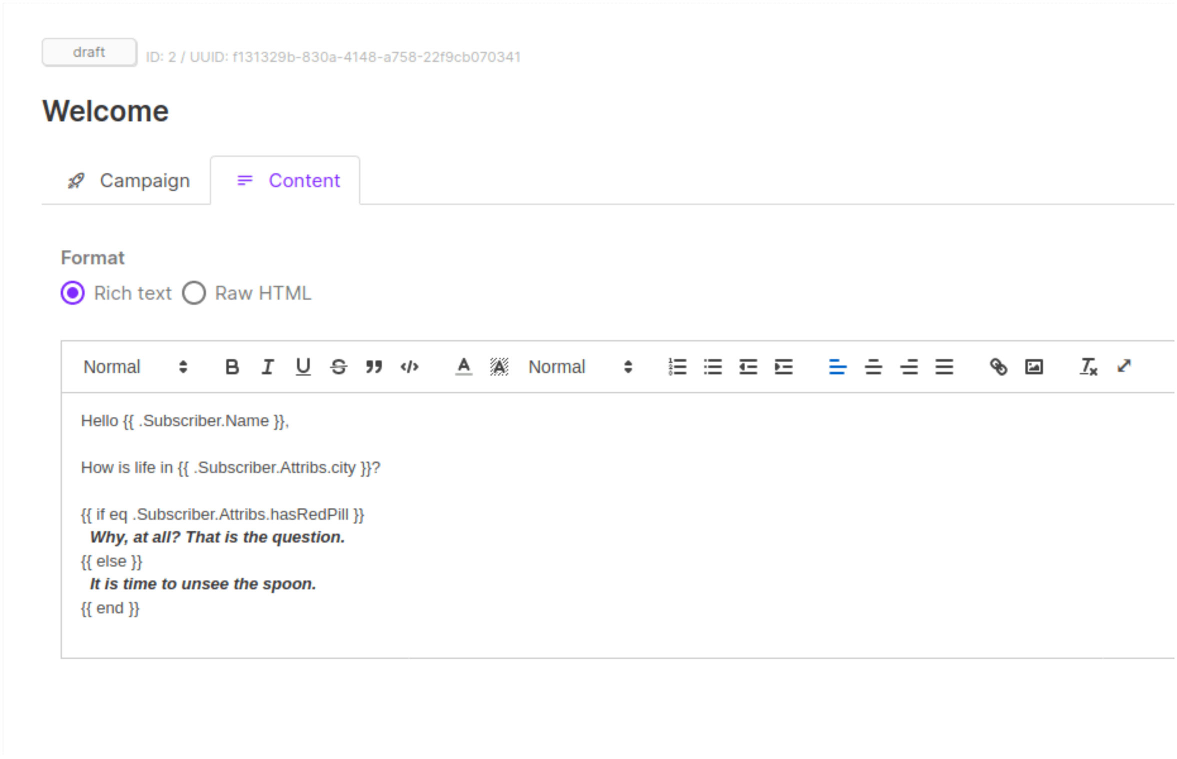
1.3.3 Listmonk

Listmonk[22] has possibly the best-designed user interface out of all mentioned self-hosted open-source mailing packages. It is written in Go[23], and paired with detailed documentation, it provides the essential tools for effective e-mail marketing – from standard recipient management and web service integration to a robust media library.

Once again, the problem with Listmonk is the editor (shown in Figure 1.4), which is more on the technical side and requires a skilled user (maybe even a developer)

20) GrapeJS `https://github.com/artf/grapesjs`
21) Mosaico `https://github.com/voidlabs/mosaico`
22) Listmonk `https://listmonk.app/`
23) Go `https://golang.org/`

to actually build an attractive and performance-driven mailing template. The editor supports the Go templating language to inject the HTML with dynamic data for each recipient (basically Golang's macros), providing functionality like the ability to address the client by name or to create conditional statements based on user information. Diverging from the best practices of a standard drag'n'drop editor with components, this platform is not usable by the regular user.



Format: [✓ Rich text] Raw HTML

```
Hello {{ .Subscriber.Name }},

How is life in {{ .Subscriber.Attribs.city }}?

{{ if eq .Subscriber.Attribs.hasRedPill }}
  Why, at all? That is the question.
{{ else }}
  It is time to unsee the spoon.
{{ end }}
```

Figure 1.4 Mailing Editor in Listmonk [6]

2 BEST PRACTICES

Webster defines best practices as a procedure that has been shown by research and experience to produce optimal results and that is established or proposed as a standard, suitable for widespread adoption. [7] In the context of UI in web applications, this pertains to component layout, color and contrast decisions, fonts and to the interface of interactive elements – which applies to both desktop and mobile design. There are also functional best practices, defining the optimal way of how a feature should work to keep the system secure, effective and intuitive.

In this thesis, the emphasis is on best practices of the actual components used in mailing platforms, like the mailing editor, formatting and styling possibilities, statistics or the predesigned templates. In the next chapters, this thesis provides a detailed analysis of each key component, its industry standards and a comparison of different solutions across the mainstream global platforms.

2.1 User Interface

By definition, user interface is a software that is designed to allow a computer user to interact with a system. [8] In other words, UI is the facade of an application with which a user interacts to control what the application does. In web applications, this is called the front-end and compared to back-end development, it is a more on the softer side of web development skills, since it involves graphic design and requires a good understanding of the end users – so that the final interface provides users a good experience. This field of expertise is called UX (User Experience) design. This part of the thesis will be looking into the ways of how certain functionality is achieved and how it interfaces with the user.

2.1.1 Mailing Editor

All existing platforms tested in this thesis use a template editor – a dynamic canvas allowing users to insert components and reorder them at will. While the components differ (as shown in Table 2.1), the core principles remain the same across all mainstream platforms. A user selects a component, adds it into the template via drag'n'drop and subsequently enters the data required. Dragging an inserted component to a different position also allows for seamless content reordering.

Table 2.1 Components In Commercial Platforms

	Mailchimp	SendInBlue	CC	Ecomail
Text	✓	✓	✓	✓
Video	✓	✗	✓	✓
Image	✓	✓	✗	✓
Divider	✓	✗	✓	✓
Spacer	✗	✓	✓	✓
Button	✓	✓	✓	✓
HTML	✓	✗	✓	✓
Social	✓	✓	✓	✓
Product	✗	✗	✗	✓

In the Table 2.1 is a set of components which could be considered the standard in e-mail marketing systems. These components work with layouts (Chapter 2.1.2) to enable multiple possibilities of content formatting.

2.1.2 Layouts

Layouts define the template structure as well as the space to insert components into. They can be full-width or multi-column, usually following the basic grid structure of Bootstrap[1]. This poses its own problem, because smaller displays cannot fit four columns in a row and have to be reformatted in a responsive manner. Figure 2.1 shows how Ecomail works with layouts.

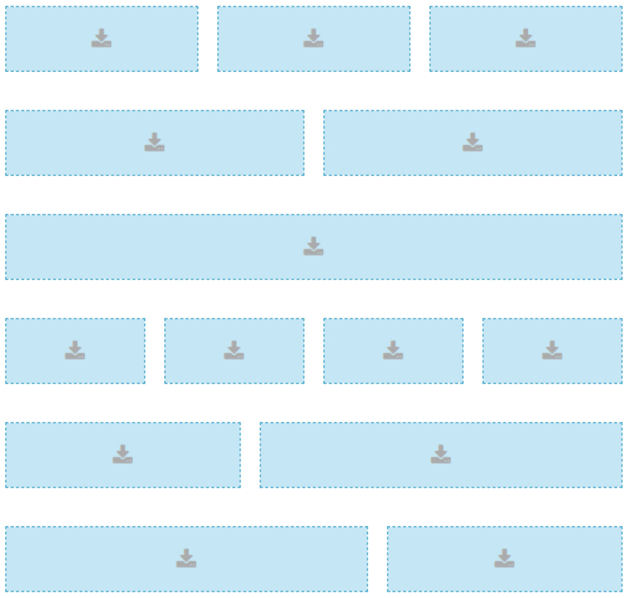


Figure 2.1 Layouts in Ecomail

[1]Bootstrap `https://getbootstrap.com/`

2.1.3 Background Styling

A mailing template does not necessarily have to be white with black text. Mainstream platforms provide an option to change background color, but some of them (Constant Contact) even offer placing a repeating pattern into the background, making the template more visually interesting. Apart from a static image, there is even a possibility to add an interactive background, which changes upon mouse hover – however, as with everything around mailing templates, only a subset of MUAs (Mail User Agent) actually support interactivity:

- Gmail / Gmail App
- iOS Mail
- Samsung Mail
- Yahoo
- AOL

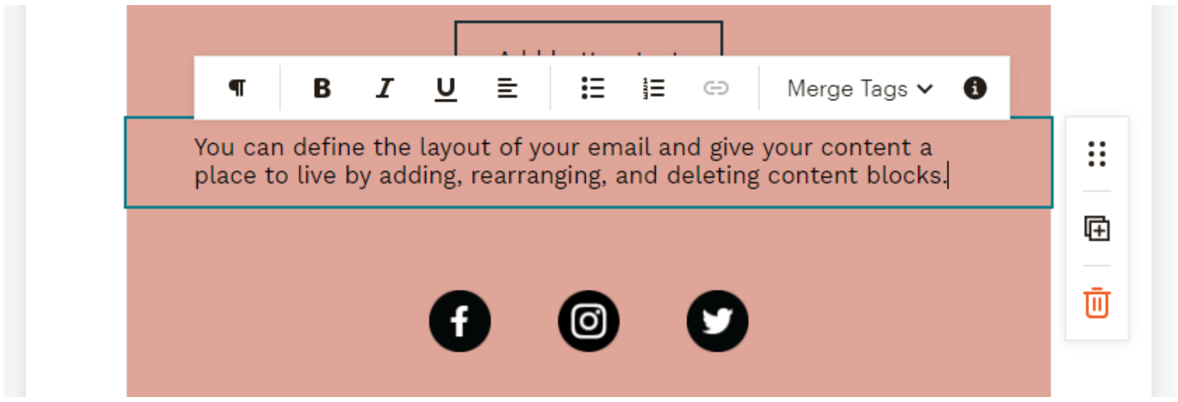
2.1.4 Text Formatting

Users expect a possibility to format text paragraphs similarly to Microsoft Word[2)] or WYSIWYG editors. For this reason, mainstream platforms provide a way to do this. The standard solution is a tooltip (a "bubble" floating above the active input field) with basic formatting options such as bold, italic, underline, paragraph align, heading, list and link. It is obvious that popular platforms toe the line of compromise – allowing users to "over-format" could result in inconsistent and unsightly mailings. The typical tooltip formatting solution can be seen in Figure 2.2.

2.1.5 Fonts

The topic of text formatting (Chapter 2.1.4) is by nature associated with fonts. Much like formatting text, users are used to the option of changing a font for headers, body or some emphasized text. Some platforms offer that option – Figure 2.3 perfectly shows the option to change font in a mailing template with a defined background pattern (more in Chapter 2.1.3) in Constant Contact template builder. As mentioned

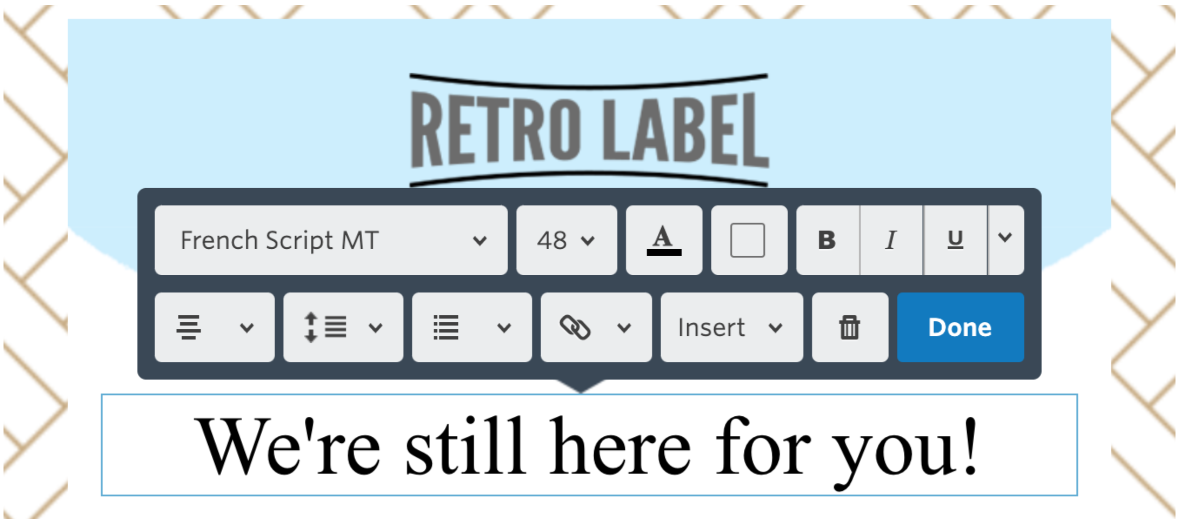
[2)]Microsoft Word `https://www.microsoft.com/en-us/microsoft-365/word`



You can define the layout of your email and give your content a place to live by adding, rearranging, and deleting content blocks.

Figure 2.2 Text Formatting In Mailchimp

in Chapter 2.1.4, there are several arguments why it is not the best idea to allow more advanced formatting options in mailing templates, so some platforms shy away from the font change option.



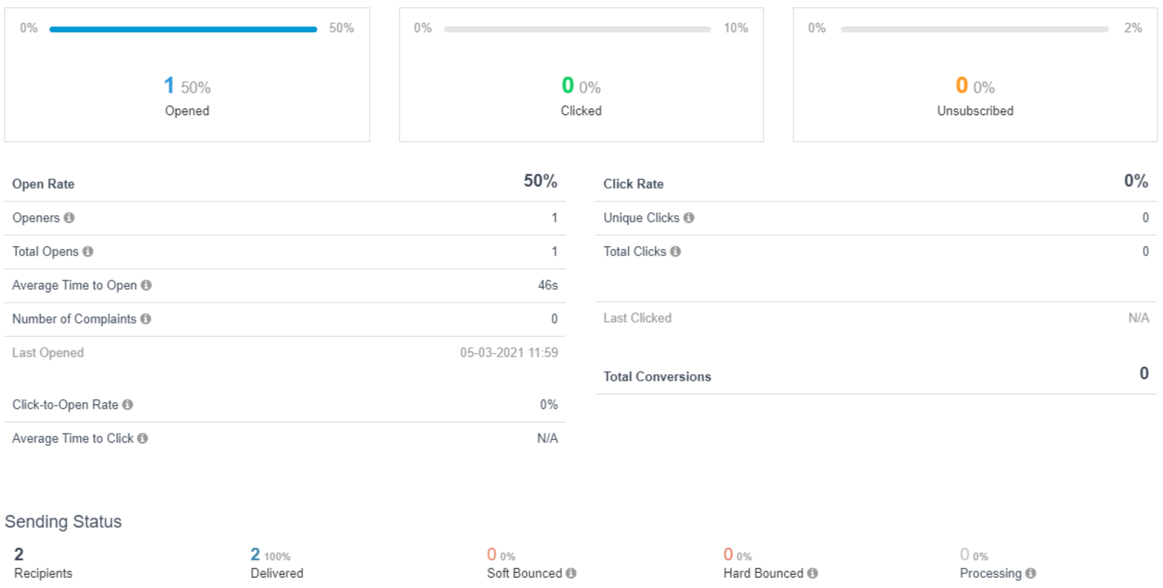
We're still here for you!

Figure 2.3 Font Change Option in Constant Contact’s Text Component

2.1.6 Statistics

Along with the mailing editor (Chapter 2.1.1), this is perhaps the most important part of the user interface. Here, users can measure the success of their endeavor and subsequently make the proper changes to better engage with their audience.

That being said, the most rudimentary mailing measurement are opened e-mails. Showing the rate of opened and unopened e-mails, this measurement can indicate if e-mails reach the recipients and if clients are even interested in the content they receive. Other metrics range from unsubscribe reasons to number of clicks for each link in the mailing. Figure 2.4 shows the statistics module in SendInBlue, which provides an outlook of the amount of opened e-mails, a sendout's clickthrough rate and the number of unsubscribes caused by a sendout.



0% – 50%	0% – 10%	0% – 2%
1 50% Opened	0 0% Clicked	0 0% Unsubscribed

Open Rate	50%
Openers	1
Total Opens	1
Average Time to Open	46s
Number of Complaints	0
Last Opened	05-03-2021 11:59
Click-to-Open Rate	0%
Average Time to Click	N/A

Click Rate	0%
Unique Clicks	0
Total Clicks	0
Last Clicked	N/A
Total Conversions	0

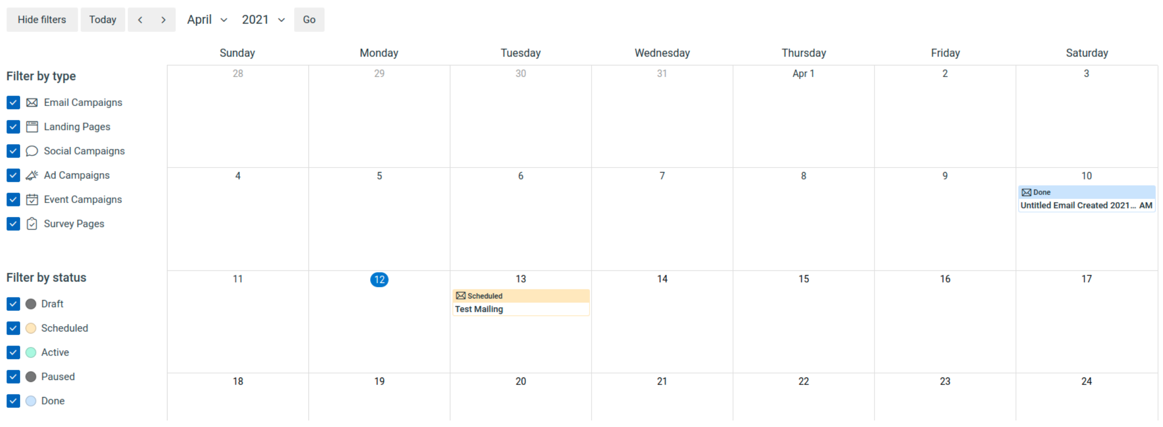
Sending Status

2	2 100%	0 0%	0 0%	0 0%
Recipients	Delivered	Soft Bounced	Hard Bounced	Processing

Figure 2.4 Campaign Statistics In SendInBlue

2.1.7 Calendar View

Mailings are usually displayed in a list view (also called a datagrid), which provides an uncluttered way of filtering and managing mailings. From the perspective of planning, however, it is far more effective to display sendout-ready mailings in a calendar view – similarly to how Google Calendar shows planned meetings. An example of this module can be seen in Figure 2.5.



Hide filters | Today | ‹ › | April | 2021 | Go

Filter by type
- ✓ Email Campaigns
- ✓ Landing Pages
- ✓ Social Campaigns
- ✓ Ad Campaigns
- ✓ Event Campaigns
- ✓ Survey Pages

Filter by status
- ✓ Draft
- ✓ Scheduled
- ✓ Active
- ✓ Paused
- ✓ Done

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
28	29	30	31	Apr 1	2	3
4	5	6	7	8	9	10 Done Untitled Email Created 2021… AM
11	12	13 Scheduled Test Mailing	14	15	16	17
18	19	20	21	22	23	24

Figure 2.5 Calendar View in ConstantContact

2.1.8 Templates

The majority of the platforms provide predesigned templates so that users do not have to start fresh each time. These templates equip the user with the ability to make more significant design changes that cannot be made through the editor. However, once a template is selected, the user has to follow through or create a new mailing with a different template.

For example, the template builder in the free version of Mailchimp offers a very small amount of 5 predesigned templates (as shown in Figure 2.6): 3 of those with white background a 2 with different background colors. Each templates has its own default component layout, placeholders and text formatting. Compared to other existing platforms, the templates in Mailchimp are rather conservative and do not provide much of additional styling.

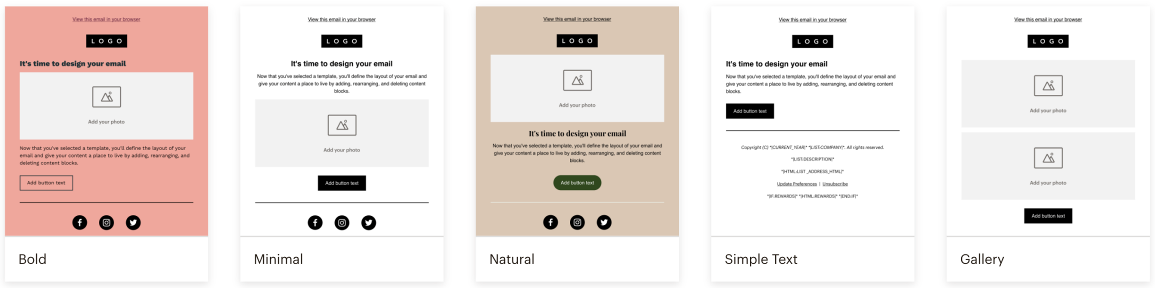


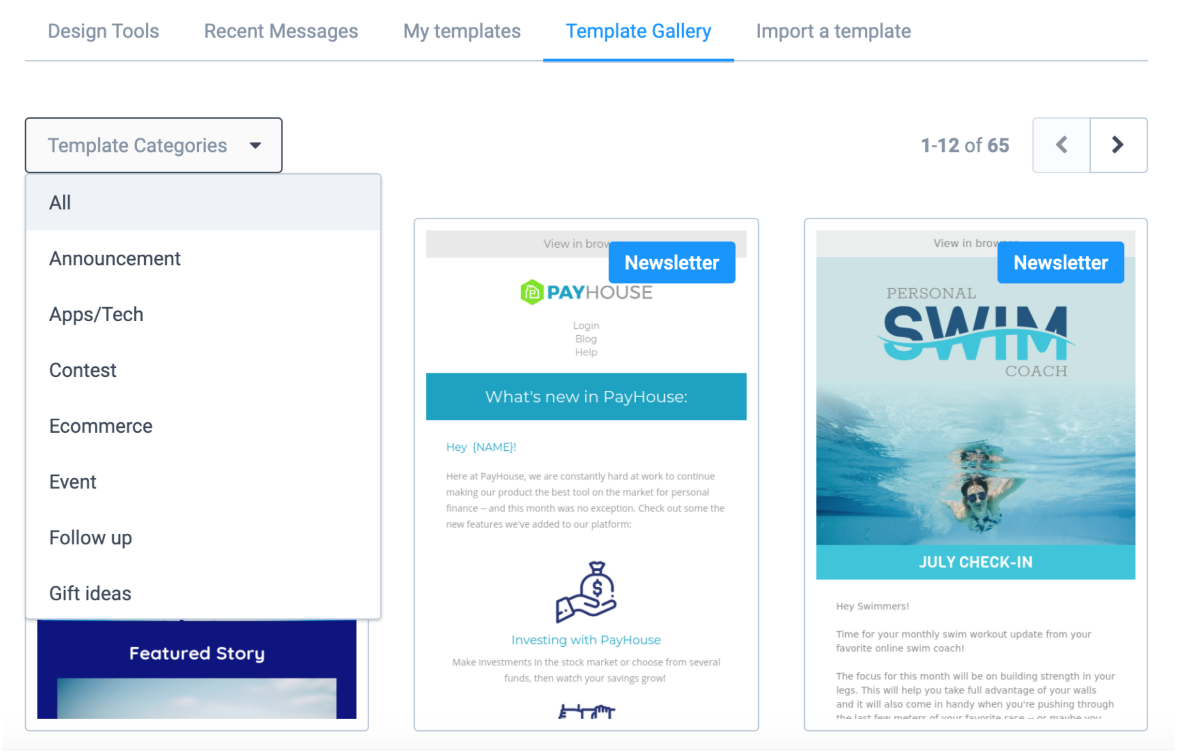
Figure 2.6 Predesigned Templates Selection in Mailchimp

On the other hand, SendInBlue has a respectable amount of 65 templates in its template gallery, grouped into special categories like event, e-commerce, transactional or webinar, which makes for easy navigation. Unfortunately, the templates are not searchable in SendInBlue, so the user has to rely on the templates being accurately categorized. These templates can be seen in Figure 2.7.

SendInBlue's templates come with prearranged sections of images, often paired with headlines matching the style of the template (as shown in Figure 2.8). In terms of templates, however, ConstantContact takes the prize with 318 searchable predesigned stylish mailing arrangements.

2.1.9 Responsive Preview

Nowadays, having the possibility to check if everything displays properly on the phone is a must. More than 70 % of people read their email in a mobile app, with most



Hey {NAME}!

Here at PayHouse, we are constantly hard at work to continue making our product the best tool on the market for personal finance -- and this month was no exception. Check out some the new features we've added to our platform:

Make investments in the stock market or choose from several funds, then watch your savings grow!

Hey Swimmers!

Time for your monthly swim workout update from your favorite online swim coach!

The focus for this month will be on building strength in your legs. This will help you take full advantage of your walls and it will also come in handy when you're pushing through

Figure 2.7 Predesigned Templates Selection in SendInBlue

checking their email in the morning. [9] That is why all the mainstream mailing platforms offer this feature (as presented in Table 1.1).

Since the creative process of building a campaign takes place on a desktop device with a bigger monitor, it is essential to allow users a preview of both mobile and tablet view. In the UI, this is usually handled via a group of buttons with icons representing different platform previews, shown in Figure 2.9. The change of platform is done dynamically, so the page does not have to reload and the user does not lose their progress – this could be achieved by either AJAX (Asynchronous JavaScript and XML) or by a more modern solution using Vue.js[3)] (more in Chapter 7.1.3).

2.2 Functionality

Having covered best practices of the UI, it is crucial to go over functionality features as well. Each mainstream platform handles them differently and in contrast to the UI, they are a little harder to analyze without some reverse engineering. Many of these functions have been adopted in this thesis and their implementation will be explained in detail further in the text.

3) Vue.js `https://vuejs.org/`

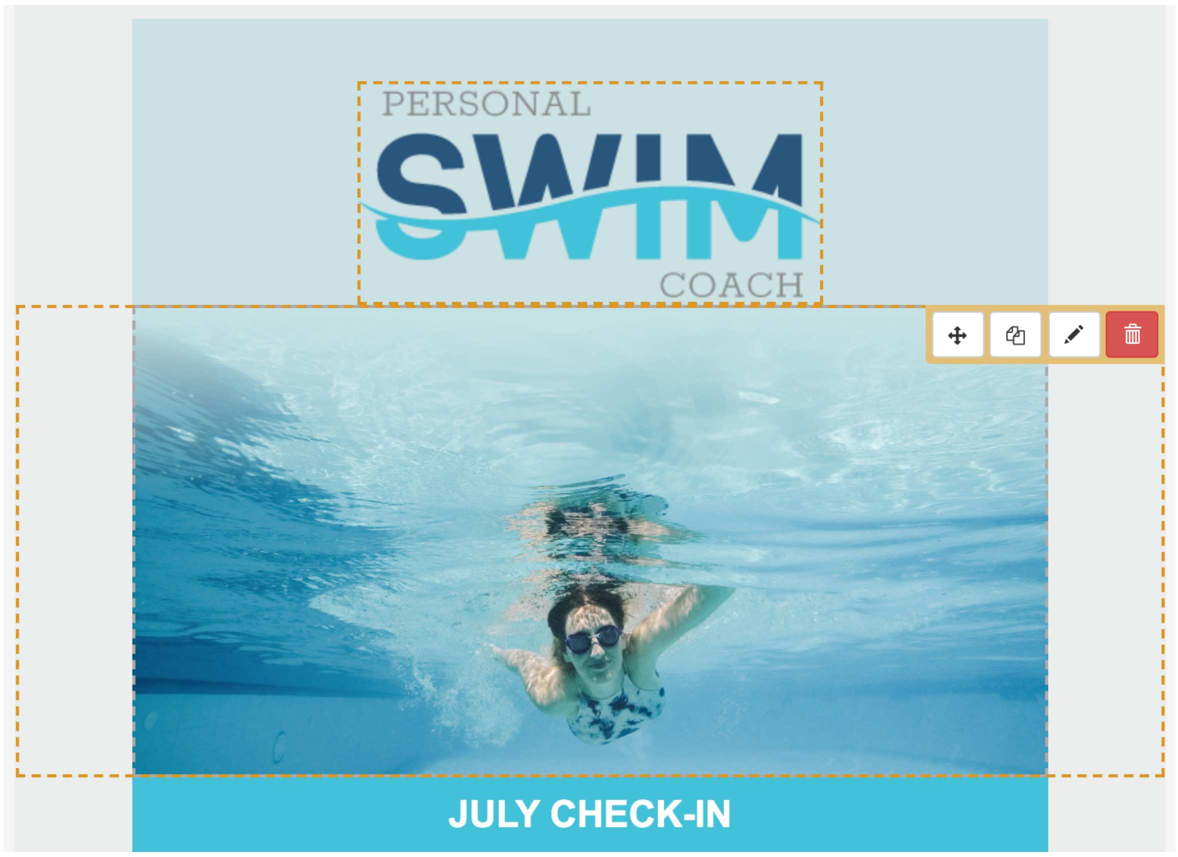


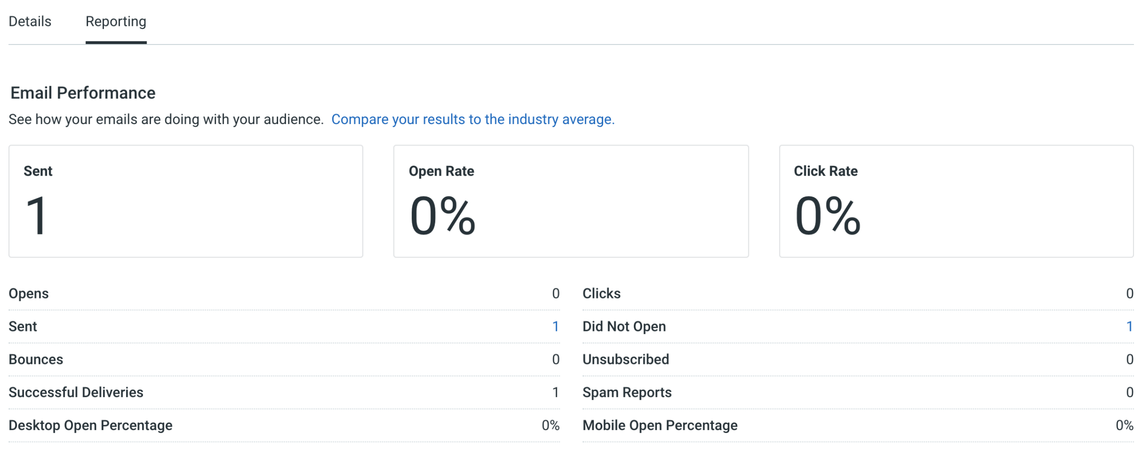
Figure 2.8 Advanced Image Layering in SendInBlue Template

Figure 2.9 Responsive
Preview Switcher

2.2.1 Conversion Tracking

Obviously, without some form of conversion tracking, a mailing platform would be essentially useless – the user has to be able to monitor the performance of each sendout, so he can make changes based on real data and the behavioral patterns of his clients. That is why all the mainstream platforms offer a reporting section, showing not only the number of recipients actually opening the e-mail, but also the clickthrough rate for each link, since both of these actions are considered a conversion in the mailing domain. The reporting dashboard of Constant Contact can be seen in Figure 2.10.

Technically, there are multiple ways of being automatically notified when a recipient opens an e-mail. Most mailing platforms use the pixel method – a 1x1 pixel image, invisible to the recipient, is inserted to the e-mail template with a specific URL in


Details Reporting

Email Performance

See how your emails are doing with your audience. Compare your results to the industry average.

Sent	Open Rate	Click Rate
1	0%	0%

Opens	0	Clicks	0
Sent	1	Did Not Open	1
Bounces	0	Unsubscribed	0
Successful Deliveries	1	Spam Reports	0
Desktop Open Percentage	0%	Mobile Open Percentage	0%

Figure 2.10 Reporting Section in Constant Contact

its source (Figure 2.11 shows Ecomail's implementation). This URL is accessed when the recipient displays the e-mail with images, prompting the e-mail client to execute HTTP requests to all images placed in the e-mail's body. The tracking URL contains data for the mailing system to be able to assign it to a specific recipient and mailing. This information then gets persisted into a database layer, while the script returns the appropriate image headers and an actual blank image. Mailchimp talks about this technique openly in its knowledge base[4]. The application created in the practical part of this thesis uses the pixel method and its implementation is described in Chapter 5.3.5.

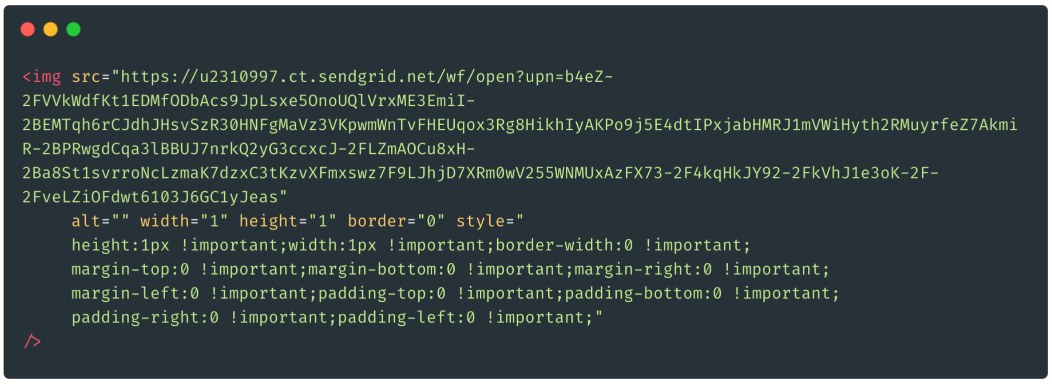

```
<img src="https://u2310997.ct.sendgrid.net/wf/open?upn=b4eZ-
2FVVkWdfKt1EDMfODbAcs9JpLsxe5OnoUQlVrxME3EmiI-
2BEMTqh6rCJdhJHsvSzR30HNFgMaVz3VKpwmWnTvFHEUqox3Rg8HikhIyAKPo9j5E4dtIPxjabHMRJ1mVWiHyth2RMuyrfeZ7Akmi
R-2BPRwgdCqa3lBBUJ7nrkQ2yG3ccxcJ-2FLZmAOCu8xH-
2Ba8St1svrroNcLzmaK7dzxC3tKzvXFmxswz7F9LJhjD7XRm0wV255WNMUxAzFX73-2F4kqHkJY92-2FkVhJ1e3oK-2F-
2FveLZiOFdwt6103J6GC1yJeas"
     alt="" width="1" height="1" border="0" style="
     height:1px !important;width:1px !important;border-width:0 !important;
     margin-top:0 !important;margin-bottom:0 !important;margin-right:0 !important;
     margin-left:0 !important;padding-top:0 !important;padding-bottom:0 !important;
     padding-right:0 !important;padding-left:0 !important;"
/>
```

Figure 2.11 Tracking Pixel in Ecomail

Albeit a less used technique, another method is Message Disposition Notifications (MDNs). This technique largely depends on the MUA, which can ignore this header by default. If the MUA respects the header, it sends a response e-mail containing the information that the message has been opened. A script can then parse the mailbox

[4]Mailchimp Open Tracking `https://mailchimp.com/help/about-open-tracking/`

and map these responses to the sendouts, basically tracking opened e-mails – achieving the same functionality as open tracker graphics without relying on recipients to allow the MUA to display images.

2.2.2 Bounce Management

Maintaining a healthy list of recipients is also an essential item in the domain of mailing system functionality. Bounces occur when an email can't be delivered to an email address. [10] There are two types of bounces:

- Soft bounce: the message is returned to sender for various reasons (e.g. full mailbox)
- Hard bounce: because an error occurred or the address is invalid, the message cannot be delivered at all

To fight against spammers, internet service providers (ISPs) monitor bounce rates for each e-mail account. Sending too many e-mails resulting in a bounce can lead to having the account blacklisted and even disabled. Bounces can happen with an out-of-date mailing list, typos in e-mail addresses or errors during recipient import.

All mainstream platforms offer automated bounce management, usually unsubscribing the bounced e-mail address. This soft-delete approach enables the user to later check for unsubscribed recipients caused by a bounce and even track bounce rates in the reporting section.

2.2.3 Recipient Rating

To provide a better overview of recipient engagement, mailing systems have a rating system in place. Every recipient is assigned a rating based on his engagement, subscribed time and bounce rate. The user can then use this rating to create segments to which he can address personalized mailings to. An example use would be an e-commerce platform rewarding their most engaged recipients with a free shipping coupon.

2.2.4 Concurrent Sendouts

Perhaps it may seem self-evident, but it is important to mention that a mailing platform should be able to function concurrently. Since businesses can operate based on engagement analysis and plan their sendouts in the most effective times, there has to be a way to send more than one mailing at one given time.

2.2.5 User Segmentation

User segmentation is basically advanced grouping of recipients. That means the user himself creates the candidate rules for each segment, mostly via a graphical representation of conditional statements. As can be seen in Figure 2.12, this interface allows users to add conditions and change the logical operator between them – for example, this segment would only include users who's MUA is Thunderbird and their locale is Czech. Created segments are then addressable as the recipient group of a mailing.

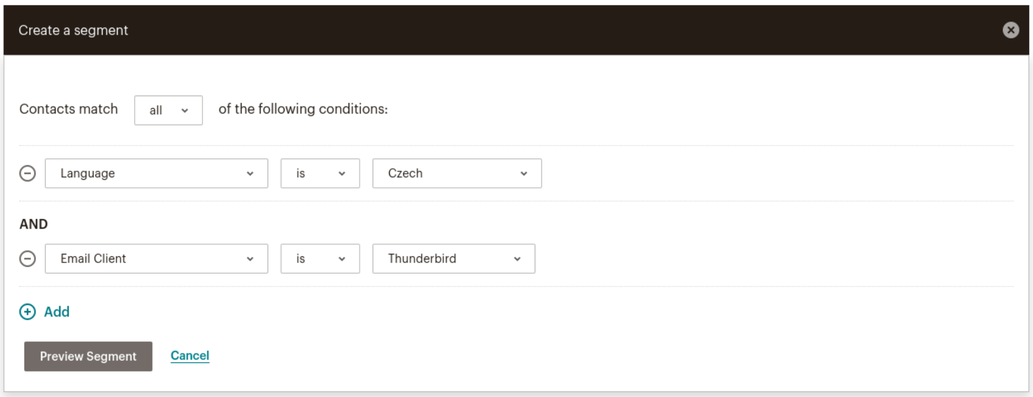


Figure 2.12 User Segmentation in Mailchimp

2.2.6 Sendout Planning

Once the mailing template is built, its recipient groups selected and its metadata filled, the user is able to either send the mailing instantaneously, or decide on a specific date and time of the sendout. This is important, take this model situation, for example: the user runs a pizza delivery business. They know that their mailing performs best if received one hour before lunch on most workdays. So that the marketing team does not have to remember to send the mailing, the user utilizes sendout planning. Since they know that it takes about 30 minutes for the sendout to their thousand-member recipient list to finish, they plans it for 10:30 AM to start from Monday to Thursday and 5:30 PM on Friday and Saturday.

3 SECURITY

In a system that is dealing with sensitive data like storing thousands of e-mails and tracking recipient behavior, it is essential to properly secure the data, the environment and the application against exploits and possible attacker entry-points. This chapter will address all the main threats when it comes to PHP web applications and what precautions this application takes to maintain the highest level of security.

3.1 PHP Vulnerabilities

As with any other web development language, PHP is also sensitive to various types of vulnerability exploits. Apart from implementing the necessary precautions explained in the upcoming chapters, it is good practice to follow these security-conscious habits:

- **Nothing is 100% secure**: someone will try something you have not thought of, or invent a new attack, and then you have to respond. [11]
- **Never trust user input**: always assume that users are out to get you, and then take steps to keep bad things from happening. [11]
- **Defense in depth is the only defense**: applying piecemeal sanitization to user input forms will just amount to a lot of code that is hard to maintain and use – however, having a function that cleans user input makes the code usable and scalable. [11]
- **Simpler is easier to secure**: if you can look at a piece of code and figure out what it does in a minute, it's a lot easier to secure it than if it takes you half an hour to figure out what it does. [11]
- **Peer review is critical to security**: a simple peer review process at regular intervals can keep bad things from happening to you and your application [11]

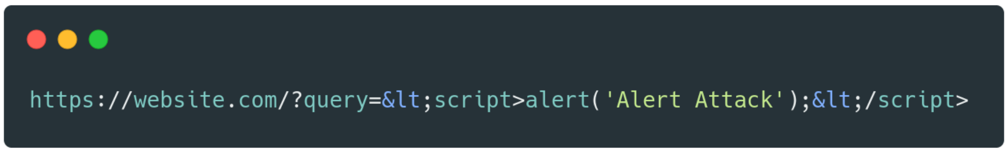
3.1.1 SQL Injection

Every time you solicit user input to construct a database query, you are permitting that user to participate in the construction of a command to the database server. A benign user may be happy enough to specify that he wants to view a collection of men's long-sleeved burgundy-colored polo shirts in size large; a malicious user will try to find a way to contort the command that selects those items into a command that deletes them, or does something even worse. [11]

Thankfully, all user input in the application created in the practical part comes from Nette Forms (more in Chapter 6.2) in `Nette\Utils\ArrayHash` format, and is later passed into objects via setter methods. These objects are then persisted with the use of EntityManager::persist() method, which automatically handles all SQL injection attempts, according to the Doctrine ORM documentation: You can consider all values on Objects inserted and updated through `Doctrine\ORM\EntityManager::persist()` to be safe from SQL injection. [12] Some parts of this application use Doctrine's `QueryBuilder`, which is also implicitly safe from SQL injection with the exception of the Expression API. This API is not used in this application, so there are no possible ways of a SQL injection attack.

3.1.2 Cross Site Scripting (XSS)

Unlike SQL injection (discussed in Chapter 3.1.1), which attempts to insert malicious SQL instructions into a database query that is executed out of public view, XSS attempts to insert malicious markup or JavaScript code into values that are subsequently displayed in a web page. This malicious code attempts to take advantage of a user's trust in a website, by tricking him (or his browser) into performing some action or submitting some information to another, untrustworthy site. [11] The attacker could contort the URL to a similar form as shown in Figure 3.1.



```
https://website.com/?query=&lt;script>alert('Alert Attack');&lt;/script>
```

Figure 3.1 XSS in URL

Nette Framework uses something called Context-Aware Escaping[1], which escapes outputted data automatically based on the current context of the code. This makes XSS virtually impossible to happen. The context check can be turned off when needed (through a Latte filter `noescape`), but developers are only advised to do so when the

[1]Latte Context-Aware Escaping `https://latte.nette.org/en/safety-first`

variable's content being outputted cannot be changed or is not supplied by the user – as mentioned in Chapter's 3.1 list of security-conscious habits: never trust user input.

3.1.3 Cross-site Request Forgery (CSRF)

A Cross-Site Request Forgery attack is that the attacker lures the victim to visit a page that silently executes a request in the victim's browser to the server where the victim is currently logged in, and the server believes that the request was made by the victim at will. Server performs a certain action under the identity of the victim but without the victim realizing it. It can be changing or deleting data, sending a message, etc. [35]

Nette Framework automatically solves this potential security problem by preventing requests being sent from another domain. This is beneficial in form submitting or Nette signals (more in Chapter 7.3.1), which automatically disallow cross-site requests.

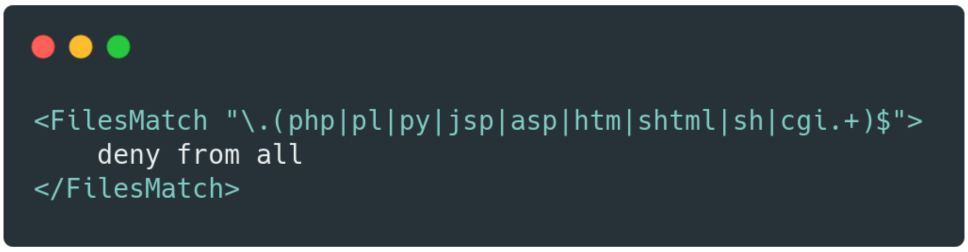
3.1.4 File Upload

Many web applications require users to upload files. This poses an issue, since files are a very dangerous user input and if not handled and validated correctly, they can lead to the application being exploited. An example of this type of attack would be uploading a PHP file and reverse engineering the upload script to understand where user files are stored. Then it is easy to execute this file by accessing it via the browser. The attacker could then change PHP's ini settings, run a delete operation on the whole directory structure or fetch the files and thus break the privacy of all data, which could mean a leak of sensitive user content.

It is essential to disallow execution of dangerous files in directories where user files are uploaded (as shown in Figure 3.2) and checking the MIME (Multipurpose Internet Mail Extensions) type of the files uploaded by users, ending all uploads of potentially dangerous files with a user error.

3.1.5 Session Hijacking

There are many types of session vulnerabilities, usually connected with stealing or spoofing a session ID, thus gaining access to the application's secured sections without actually being authenticated. This can be fought by setting the right configuration



```
<FilesMatch "\.(php|pl|py|jsp|asp|htm|shtml|sh|cgi.+)$">
    deny from all
</FilesMatch>
```

Figure 3.2 Disallowing Dangerous Files in .htaccess

in PHP's ini file, however, Nette Framework does this automatically – provided the `ini_set()` directive is allowed.

3.2 Password Hashing

The application uses Nette Passwords[2)] for hashing and verifying passwords. This class uses PHP's `PASSWORD_DEFAULT` method with the algorithmic cost of 12 (unless configured differently), and can be passed to other classes via dependency injection, unlike its older version which could only be used statically.

Behind the scenes, the `PASSWORD_DEFAULT` constant currently defaults to `BCRYPT`, which is a password hashing algorithm and it is not the same as just encryption in general. It is used specifically for encrypting and securely storing passwords. It is used primarily when a user enters a password and that password needs to be stored in a database in a way that the original password could not be guessed even if the system was attacked and the database got compromised. [13]

3.3 GDPR

The General Data Protection Regulation (GDPR) is the toughest privacy and security law in the world. Though it was drafted and passed by the European Union (EU), it imposes obligations onto organizations anywhere, so long as they target or collect data related to people in the EU. The regulation was put into effect on May 25, 2018. The GDPR will levy harsh fines against those who violate its privacy and security

[2)] Nette Passwords `https://doc.nette.org/en/3.1/passwords` standards, with penalties reaching into the tens of millions of euros. [14]

When discussing GDPR security in the context of mailing systems, it is important to realize that the transactional URLs sent within the e-mail template can be abused. To unsubscribe a recipient, for example, the common way would be to send a link structured like this: `example.com/unsubscribe/example@gmail.com`. It is obvious where the problem lies now – the attacker can write a script looping through an e-mail dictionary, send request to this URL and monitor server responses. If the server returns a 404 code, it can be inferred that this person is not in the recipient database, and if the server returns a 200 code, the attacker knows this e-mail address exists in the database. Even if the server always return a unified response code, the resulting HTML can be parsed and understood by the attacker as well.

That is why this application never sends transactional links containing any personal information which could be exploited. Instead, every link contains a Queue object hash, which allows the application to identify the entities involved: e.g. the recipient who is trying to unsubscribe. This hash cannot be reverse engineered and thus the unsubscribe link is safe from this type of attack, which would lead to a private data leak.

II. PRACTICAL PART

4 MODELS

A model represents the underlying logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface. [15] In this chapter, the thesis will cover the model representation used in this application – specifically Doctrine entities and their respective repositories.

4.1 ER Diagram

The Entity-Relationship Diagram (ERD) for this application can be found in Appendix 1. It maps the attributes of each entity and pairs the foreign key indices to related entities, showing the cardinality of both sides of the relationship, using the crow's foot representation (shown in Figure 4.1 and 4.2). There are also three types of ER diagrams: conceptual, logical and physical. The one provided in Appendix 1 is a physical ERD, since it provides column types, primary keys and foreign keys.

Figure 4.1 show the one-to-many relationship between the Element entity and the Conversion entity. One element can have multiple conversions.

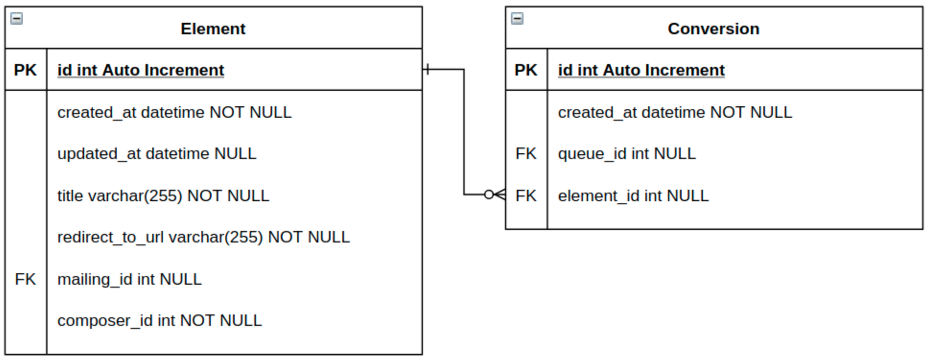


Figure 4.1 One-to-Many Relationship

Figure 4.2 shows the many-to-many relationship between the Recipient entity and the RecipientGroup entity. Many recipients can have many groups they belong to. This is represented by a junction table, which is then automatically joined by Doctrine upon receiving any queries involving recipients and their groups.

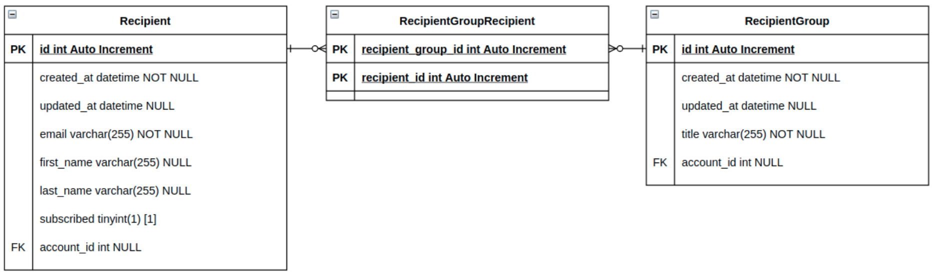

Recipient
PK id int Auto Increment
created_at datetime NOT NULL
updated_at datetime NULL
email varchar(255) NOT NULL
first_name varchar(255) NULL
last_name varchar(255) NULL
subscribed tinyint(1) [1]
FK account_id int NULL
RecipientGroupRecipient
PK recipient_group_id int Auto Increment
PK recipient_id int Auto Increment
RecipientGroup
PK id int Auto Increment
created_at datetime NOT NULL
updated_at datetime NULL
title varchar(255) NOT NULL
FK account_id int NULL

Figure 4.2 Many-to-Many Relationship

4.2 Use Case Diagram

Since mailing platforms are role-driven systems, the key attribute of back-end business logic is the user role. This dictates the user's capabilities. A standard user is able to manage recipients, recipient groups and mailing. They are also allowed to switch accounts (provided they have more than one), view sendout statistics, import recipients into recipient groups, change mailing status and send test sendout of a mailing. The administrator role inherits the standard user permissions and extends them with user and account management capabilities.

In this thesis, management of an entity means browse, read, edit, add and delete rights – which is known as BREAD, but also as DAVE (Delete, Add, View, Edit), CRAP (Create, Replicate, Append, Process) or CRUD (Create, Read, Update, Delete). The different capabilities of all system user roles can be found in the Use Case diagram (Figure 4.3).

4.3 Entities

Drawing from the diagrams presented in chapters 4.1 and 4.2, the application's database is generated automatically based on the schema defined in Doctrine entities. This chapter lists all entities found in the system, detailing their contents.

4.3.1 Account

It is imperative to start with the Account entity, as it allows users to have multiple dashboards to manage while keeping all settings, statistics and mailings completely separate. This way, each account can have its own SMTP configuration and recipients, thus resulting in unique statistics reports. Apart from separate SMTP configuration, the Account entity provides an option to select a primary and secondary color to style

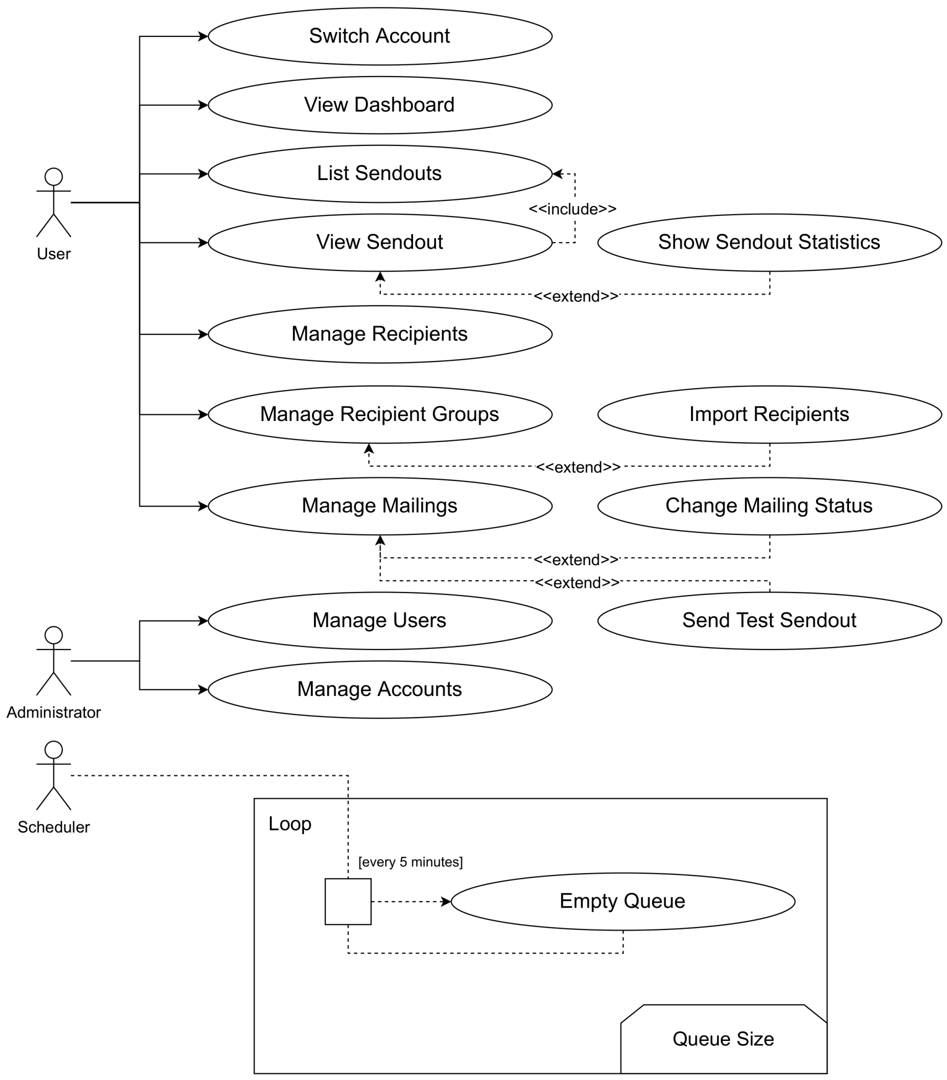


Figure 4.3 Use Case Diagram

the mailings in the design of the account brand. Users can also upload a logo that is shown in the mailing's header.

4.3.2 User

The User entity allows the application to house many different users, store their credentials, roles and their owned Accounts. A User's status and role is mapped via a class constant rather than a standalone entity, since there is no need to add new roles and states from the user side. A User can have two roles: administrator or user. An administrator can manage Users and Accounts, while a basic user can only manage entities that pertain to mailings only. User status was implemented in case there is a need for user registration from the front-end. A user would then have to be activated before they can sign in.

4.3.3 Recipient

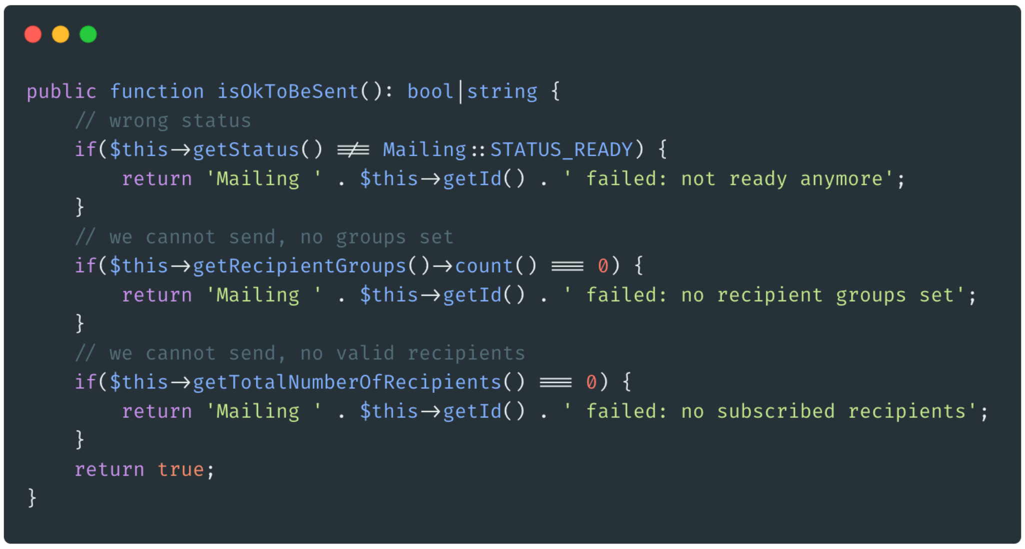
To complete a mailing blast, the application needs a set of subscribed recipients to send to. It is possible to fill out their name and surname, but the only required field is the recipient's e-mail – along with pertinence to different recipient groups and a Boolean flag to mark if the user is still subscribed.

4.3.4 Recipient Group

To be able to target mailings on various unique segments of recipients, it must be possible to create recipient groups. These just serve as a list of names with clear demarcation of user accounts they belong to.

4.3.5 Mailing

Perhaps the most important entity in the system, the Mailing entity, represents a mailing users build in the editor (Chapter 2.1.1). This entity has a set of states it can be in, with the first one being a concept. Once a user sets the status to ready, the application fetches the mailing inside of the sendout script and decides if it should be sent. That only happens if specific conditions (shown in Figure 4.4) are met.



```php
public function isOkToBeSent(): bool|string {
    // wrong status
    if($this->getStatus() !== Mailing::STATUS_READY) {
        return 'Mailing ' . $this->getId() . ' failed: not ready anymore';
    }
    // we cannot send, no groups set
    if($this->getRecipientGroups()->count() === 0) {
        return 'Mailing ' . $this->getId() . ' failed: no recipient groups set';
    }
    // we cannot send, no valid recipients
    if($this->getTotalNumberOfRecipients() === 0) {
        return 'Mailing ' . $this->getId() . ' failed: no subscribed recipients';
    }
    return true;
}
```

Figure 4.4 The method deciding if a mailing should be sent

Another important part of this entity is the attribute `jsonData`, which houses all data from Vue.js sent from the editor. This data is later interpreted into Latte templates, which are compiled into HTML and sent to the recipient.

4.3.6 Element

To provide statistics not only on e-mail opening but also on click-through rates of different links in the e-mail itself, it is necessary to route these links through the application. To achieve this, once a user creates a link inside the editor, its database counterpart gets created immediately and generates a link to replace the original one. This way, a recipient click the rerouted link pointing to the application's URL – there, a visit is recorded into statistics and the user is redirected to the original link destination.

4.3.7 Conversion

As mentioned in the previous section, the entity Element is the application's way of keeping track of recipients' click-through rate. Usually, this is the ultimate goal of mailings and thus could be called a conversion. Obviously, the actual conversion only happens after the recipient not only visits the website, but buys a product or sends an inquiry – still, the client does this based on the conversion inside of the mailing itself.

The entity Conversion saves every click-through of the component Button, saving the e-mail of the recipient completing the transaction and the Article where the link resides. This way, an administrator can view how different buttons perform in a single Article and make informed placement decision in the future.

4.3.8 Queue

The Queue entity is used in the `SendMailingsCommand` (more in Chapter 4.5.1) script, where it is initialized and filled with recipients for each Article getting sent. The queue is then emptied while maintaining a status to be able to distinguish sent and unsent rows – this is achieved by a simple Boolean flag. The same applies for the information if the e-mail was opened or not.

4.3.9 Sendout

This entity serves as a record of individual sendouts of each mailing. A mailing can be sent multiple times, since clients usually want to test if the mailing displays properly on all devices before sending it to the production recipient database. The application notes the time the sendout started and when it ended, indicating the total time it took for the sendout script to dispatch the queue.

4.3.10 Unsubscribe

It is possible for a client to choose if an unsubscribing recipient is required to enter a reason of his opt-out. That is why the Unsubscribe entity exists. Otherwise, it would be sufficient to only toggle the subscription flag in the recipient record directly.

4.4 Repositories

To work with Doctrine entities, the developer can call the entity manager to retrieve an entity repository via its class name. Repository is a design pattern used to implement DAL (Data Access Layer) along with DAO (Data Access Objects) and is ideal for methods fetching a collection of objects or adding some new entity. With Doctrine, a repository is a class extending `EntityRepository` – in this application, the generalization is furthered by having a base abstract class `AbstractRepository` from which all project repositories inherit.

4.4.1 Element Repository

This repository is mainly used to create an Element entity out of a button component. The method `ElementRepository::fromButton(<array>)` takes an array variable received from the mailing editor via the internal API, checks the necessary attributes and persists a new Element record in the database. If the entity is persisted successfully, the entity is returned; otherwise the method returns null. This method is used to replace the actual URL of a button with a special link which allows the application to track the clickthrough rate of each button component. Upon visiting this link, the visit is noted in the database and the recipient is then redirected to the original destination specified in the editor.

4.4.2 Mailing Repository

Since this application works with user accounts which can be switched arbitrarily, it is necessary to fetch entities with respect to the account designation. Mailings follow the same constraints, and to adhere to the DRY (Don't Repeat Yourself) principle, the method `MailingRepository::getByAccount(<int>)` takes the account ID and returns the `QueryBuilder` object. This way, the query can be specified further in each context, before the results are fetched.

4.4.3 Recipient Repository

Upon sending a mailing, all recipients are paired with the mailing to be sent and pushed into a sendout queue. This is done via the Queue entity, which maps the recipient to his e-mail to speed up the sendout algorithm – the script then only has to access a string object, instead of a whole Recipient entity. This creates a problem when a recipient wants to unsubscribe: for security reasons (more in Chapter 3), the only identifier sent inside the actual mailing is a hash, identifying a Queue entity.

Once the application has the Queue entity of the recipient, it has to somehow find the recipient with that e-mail and unsubscribe him. The important part here is that the application has to unsubscribe this recipient only for the account of the mailing which prompted the opt-out. Otherwise, the recipient would be unsubscribed from all accounts in the database. For these reasons, `RecipientRepository` has a method `findByQueue(<Queue>, <bool>)`, which is used in the unsubscribe API endpoint to find either a subscribed recipient from an account, or an unsubscribed recipient from an account – since the recipient has the option to resubscribe after filling the opt-out form.

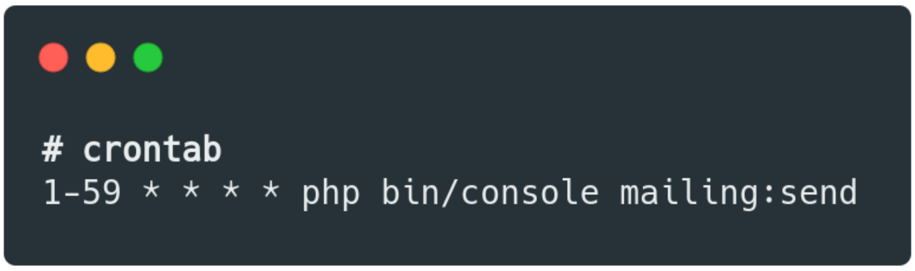
4.4.4 User Repository

The only important method in this repository returns a User entity object or null, if no results are found. The method `UserRepository::findOneByEmail(<string>)` is used in the `UserAuthenticator` class, which handles the authentication of a sign-in request.

4.5 Commands

Usually, bigger clients require thousands of recipients segmented across multiple recipient groups. Consequently, it is not possible to send mailings that take more time to completely send than a standard PHP timeout directly in the browser. It could be done in 60 second batches via an HTTP cron job, but that is a fairly unreliable and outdated solution.

Nowadays, it is easy to interface with a PHP application via the CLI (Command Line Interface), which provides a virtually unlimited timeout. For this reason, it is a good idea to write scripts meant to be run periodically as console commands. An ideal tool for this is Symfony Console (more in Chapter 7.2.3), which creates a wrapper offering the programmer an easy interface with a single execute method with access to both input and output. Thanks to that, console commands can be decorated with arguments and parameters resembling classic bash commands. Figure 4.5 is an example of a crontab entry that makes the application check the queue every minute.



```
# crontab
1-59 * * * * php bin/console mailing:send
```

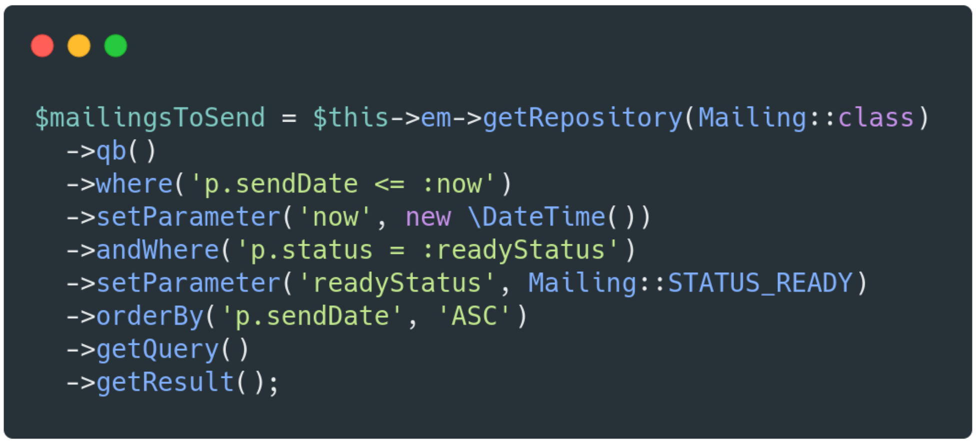
Figure 4.5 Crontab Entry for Mailing Sendout

4.5.1 SendMailingsCommand

This command is possibly the most indispensable section of code in the whole application, since it takes care of the actual sending of e-mails. The command optionally receives a test flag, but is run without it in production Crontab, as shown in Figure 4.5. This could have been done with separate methods in a presenter or as a helper function in a repository, but since there are fewer limitations in CLI, it has been written as a console command. The command uses a rather simple algorithm which handles a set of various tasks, shown in the list below:

1. Get all mailings that are ready to send
2. For each mailing, set that status to sending
3. Push each recipient of the mailing into the queue
4. Create a new Sendout instance
5. Start emptying the queue and for each record, either:
    - If the application is running in development mode, output a debug string
    - If the application is running in production mode, send the e-mail

As mentioned in the item 1 of the list above, it is imperative that the application fetches correct mailings to be sent. The query in Figure 4.6 returns only those mailings that are actually ready and not planned to be sent in the future.



```php
$mailingsToSend = $this->em->getRepository(Mailing::class)
  ->qb()
  ->where('p.sendDate <= :now')
  ->setParameter('now', new \DateTime())
  ->andWhere('p.status = :readyStatus')
  ->setParameter('readyStatus', Mailing::STATUS_READY)
  ->orderBy('p.sendDate', 'ASC')
  ->getQuery()
  ->getResult();
```

Figure 4.6 Algorithm Fetching Mailings Ready for Sendout

5 MODULES

Nette Framework internally allows structuring the application into modules, presenters and actions. A typical use of this feature would be having a Front module and a CMS module, logically separating the two different environments. This separation then allows for different user storage and different authentication – so the Front authenticator can handle sign-in requests of users in the front-end (like clients of an e-shop accessing their account), and the CMS authenticator can handle administrators accessing the content management system. Also, modules make creating internal links much more organized. Inside a module environment, Nette defaults to the current module unless explicitly specified, so a link like `Presenter:action` is enough. However, if a link should traverse to another module, it can be specified as `Module:Presenter:action`. The structure of the application needs to adhere to these principles, with some room for customization – more on this in Chapter 8.1.

5.1 Introduction to Presenters

Nette Framework is built upon the MVC (Model-View-Controller, more in 7.3.3) design pattern with some slight changes – for example, controllers are called presenters. A presenter fetches data from models and provides it to a view template. Technically, presenter is a class that represents a specific page of a web application, such as homepage, product or sign-in page. A presenter's task is to process a request and return a response (which can be an HTML page, JSON or even a redirect). [16]

In Nette, presenter is any class that implements the class `Application\IPresenter`. A presenter has its own life cycle, consisting of methods called in a specific order, as shown in Figure 5.1. These methods do not need to be implemented unless there is a need to override them, e.g. for checking if the user is authorized to view the page in the `startup()` method or to send specific variables to the template in the `render<View>` method.

There is a small semantic difference between `action<Action>` and `render<View>` methods, which this application utilizes. Since the action method runs before the render method, it makes sense to fetch all the data from models and prepare them into class attributes, which are then accessible in the render method where they just get injected into the template. Another important feature of a presenter is signal handling, which is principally used for asynchronous calls from AJAX. Chapter 7.3.1 visits this topic in detail.

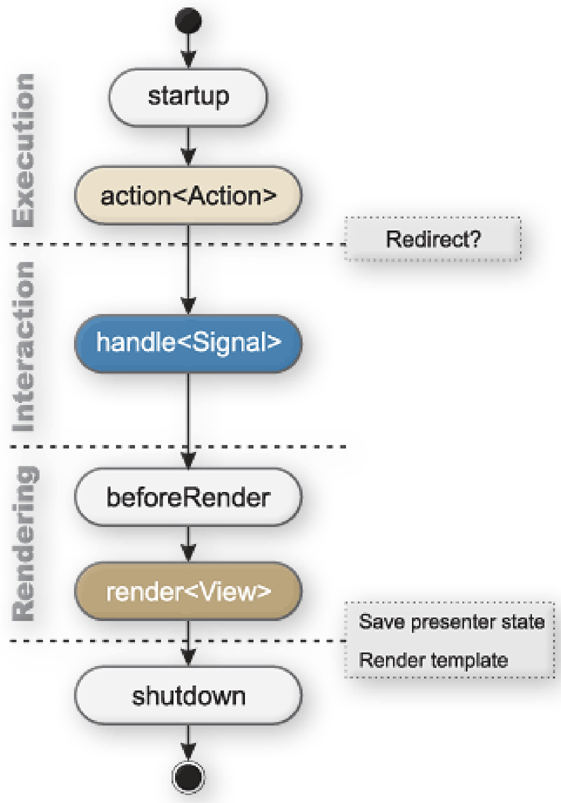


Figure 5.1 Nette Presenter Life Cycle

5.2 Front Module

In this application, there is basically no module completely accessible without the user being authenticated. Commonly, the Front module would be public, showing the client-side of a web application. However, to be able to use the functions of this system, the user has to be authorized to make changes in the system, which is only possible if the user is authenticated.

5.2.1 Secured Presenter

All presenters are classes implementing the `Application\IPresenter` interface and can extend any classes meeting this constraint. This inheritance is useful, since many presenters require the same services from the DI (Dependency Injection) container. Also, it is possible for the inherited abstract presenter to provide common variables (like the website title, for example) to the template. For this reason, all Front module presenters extend `SecuredPresenter` to some degree, because it checks if the user is authenticated and implements some basic functionality necessary for all other presenters.

5.2.2 Sign Presenter

This presenter, even though it extends Secured Presenter (Chapter 5.2.1), does not require the user to be authenticated. If it did, there would be an endless cycle of redirects. That is why `SignPresenter` actually overrides the `checkRequirements()` method, where the authentication check resides in the inherited presenter. The important fact to mention here is that the session section `account` has to be purged with each sign-in or sign-out, so that the user does not spoof the session data with different account's ID, or, if the account was deactivated, the account's ID is not stuck in the session data.

5.2.3 Home Presenter

`HomePresenter` is virtually the application's dashboard. It only has one action, fetching various statistics data and condensing it into an `ArrayHash` variable, which is then sent to the template. The template then uses Chart.js[1] to present the data in graphs and charts.

5.2.4 Datagrid Presenter

This is an abstract presenter, which initializes the `QueryBuilder` and applies all filter and sort rules based on inherited persistent[2] class attributes. It also contains the methods which change the sort type and direction, or the filter query and field. Datagrid presenter also houses the `handleChangeStatus` method, which provides the functionality of switching a status type to a different value, and the `handleToggleField` field, working similarly but only in Boolean context, toggling a field to a desired bit value.

5.2.5 Mailing Presenter

This presenter extends the class `DatagridPresenter`, and is a typical list-view control node, sending an array of Mailing entities to be listed in the datagrid. It also creates the `SendTestForm` component, which allows users to provide a test e-mail address to instantly send a preview of the mailing. Also, since the Editor presenter (Chapter 5.2.8) often persists empty entities, it is essential that the list does not show empty records with no data filled. This is achieved in the template with a Latte macro called `skipIf`.

[1] Chart.js `https://www.chartjs.org/`

[2] Nette Persistent Parameters `https://doc.nette.org/en/3.1/components`

5.2.6 Recipient / RecipientGroup Presenter

These presenters not only provide the data to list the entities in datagrid view, but also automatically fill the entity form if an existing record is being edited. This is achieved with the method `actionForm(<?int>)`, which can receive an ID of the record to be edited. If no ID is supplied, the application assumes the request is to create a new entity. This principle is repeated in all presenters that create an entity form component.

5.2.7 Sendout Presenter

Apart from the list-view data fetching, this presenter assembles the statistics data for the sendout detail template, which shows a chart representing the rate of opened / unopened e-mails. Another information this presenter shows is the click-through rate of each Element entity present in the mailing of the sendout.

5.2.8 Editor Presenter

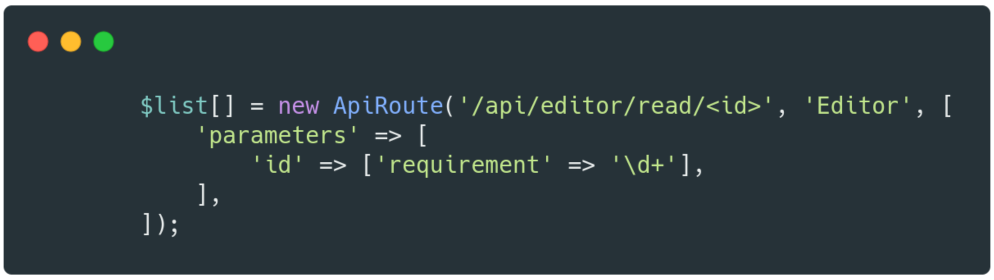
`EditorPresenter` houses an interesting algorithm – since the Vue.js editor (more in Chapter 7.1.3) works with progressive saving and can be essentially left unfilled, it is important to first create the entity that all the entered data will pertain to. If no ID is provided to the `EditorPresenter::actionDefault(<?int>)` method, the script automatically creates a new empty Mailing entity and redirects the user to the same method, but now with an ID. The user does not notice any unusual behavior. Also, since the user is able to request a preview of the mailing template he built in the editor, the method `EditorPresenter::actionPreview(<int>)` handles this request and supplies the HTML of the Mailing entity to the template.

5.2.9 Unsubscribe Presenter

This presenter is yet another instance of a presenter overriding the `checkRequirements()` method so that it is accessible without authentication. This is because the presenter's templates need to be viewed by a recipient who will not have an account in the mailing system and thus cannot sign-in. First, the recipient is identified by a combination of Queue hash, Mailing ID and also an Unsubscribe ID (which is created in the endpoint one step before – more in Chapter 5.3.7). The presenter then automatically fills the opt-out form (if shown) with the recipient data. If the recipient made a mistake and wants to resubscribe, he is given a choice with a button that displays after a successful opt-out. The algorithm then finds the Unsubscribe entity of that recipient, changes the Boolean value of the resubscribe field and amends the subscribed flag in the Recipient entity.

5.3 API Module

Often times, modern applications need a way of communicating between front-end and back-end – in this case, PHP with Javascript. In contrast to Laravel[3], Nette does not provide API features out of the box, so it is beneficial to use a community package supplementing this behavior. In this application, the package contributte/api-router is used to create RESTful API routes. In this context, RESTful means the developer can specify which HTTP method will the endpoint support, and handle the supplied data accordingly. This can be done either with annotations or inside Nette Router. Figure 5.2 is an example of this routing.



```
$list[] = new ApiRoute('/api/editor/read/<id>', 'Editor', [
    'parameters' => [
        'id' => ['requirement' => '\d+'],
    ],
]);
```

Figure 5.2 API Router Implementation

Essentially, these endpoints are just standard presenters with predefined methods like `actionRead`, `actionCreate` and so on. The Contributte package then makes sure that a GET request is sent to `actionRead`, POST request to `actionCreate`, etc. In the following chapters, there are examples of the endpoints and their purpose.

[3]Laravel `https://laravel.com/`

5.3.1 Editor Endpoint

Starting with the most important endpoint, the `API\Editor` class, this endpoint supports a POST and a GET HTTP request. With a GET request, the endpoint receives a JSON string from Vue.js containing the editor contents, followed with a Mailing ID. Once decoded from JSON, the endpoint checks if the data is consistent and ready to be persisted. Apart from editor's components, the mailing meta data like the title or the subject are parsed and saved into the database.

5.3.2 Conversion Endpoint

As briefly mentioned in Chapter 3 detailing the application's security measures, the actual recipient's e-mail is not present in any links inside the mailing – instead, the sendout algorithm creates a unique hash for each Queue entity, which is later used to identify the recipient internally. A request to the link's destination is handled by the API module, so when the recipient clicks any button link, they are sent to the method `API\Conversion::actionRead`. This method receives two strings: a Queue hash and the ID of the clicked Element entity. This gives the application enough data to create a new Conversion entity, create the correct relationships and persist it – but only for the first click, so an attacker cannot overflow the database with bogus conversions. After that, the recipient is redirected to the original URL of the button.

5.3.3 Element Endpoint

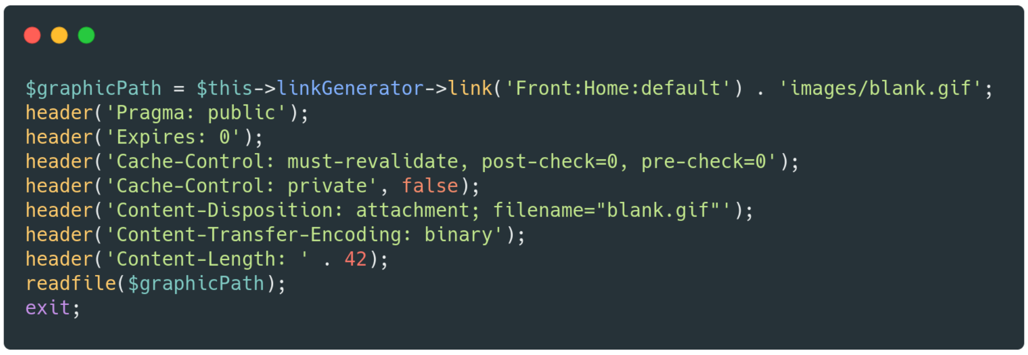
This is an internal API endpoint which handles the creation of an Element entity from a button using a method detailed in Chapter 4.4.1. Upon successful entity persistence, the endpoint returns the link to replace the original one. This link is then sent inside the e-mail template and delivered to the recipient. If clicked, the endpoint described in Chapter 5.3.2 handles the redirect to the original link.

5.3.4 Image Endpoint

Another internal API endpoint, `API\Image`, works with the Vue.js editor to upload a user-provided image. In the HTTP POST request, Vue.js sends base64 encoded image data. The endpoint's role is to convert it to an object of the `Utils\Image` class and save it to the server, returning its path to Vue.js, which then displays it in the editor.

5.3.5 Queue Endpoint

Perhaps a more interesting endpoint than others, this public endpoint handles the monitoring of mailings being opened. As detailed in Chapter 2.2.1, this application uses the pixel method to track opened e-mails. This is achieved by inserting an image with a URL with this endpoint as a destination to the end of the e-mail template. When the recipient accesses it upon displaying all images in the e-mail (basically opening it), the endpoint is sent the Queue hash and the Mailing entity ID. With this information, the application is able to change the flag `opened` in the Queue entity to true. After persisting the edited entity, the endpoint returns a binary response, as show in Figure 5.3.



```php
$graphicPath = $this->linkGenerator->link('Front:Home:default') . 'images/blank.gif';
header('Pragma: public');
header('Expires: 0');
header('Cache-Control: must-revalidate, post-check=0, pre-check=0');
header('Cache-Control: private', false);
header('Content-Disposition: attachment; filename="blank.gif"');
header('Content-Transfer-Encoding: binary');
header('Content-Length: ' . 42);
readfile($graphicPath);
exit;
```

Figure 5.3 The Pixel Method Implementation

5.3.6 RecipientGroup / SelectedRecipientGroup Endpoints

These two internal endpoints help the Vue.js editor to work with recipient groups. While the `API\RecipientGroup` endpoint returns all possible groups for the selected account, the `API\SelectedRecipientGroup` returns already selected recipient groups for a mailing. First, the Mailing entity is found via an ID sent as a parameter of the GET request. The endpoint's algorithm then finds the correct entity and returns the result formatted as a JSON array, containing the ID and title of each selected `RecipientGroup`.

5.3.7 Unsubscribe Endpoint

`API\Unsubscribe` is a very important endpoint, since it provides the functionality of opting out of receiving mailings, which is legally required to send marketing e-mails. The method `API\Unsubscribe::actionRead()` takes a Queue hash and a Mailing ID to identify the recipient to unsubscribe. Using the `RecipientRepository`'s method `findByQueue` (described in Chapter 4.4.3), the script unsubscribes all the found recipients and sets their subscribed attribute to false. If the account of the mailing has a set unsubscribe redirect URL, the recipient is then redirected to that URL. Otherwise, the recipient is redirected to a default Unsubscribe template – detailed in Chapter 5.2.9.

5.4 Admin Module

The application offers a separation of user roles, defining a user and an admin role. Admin users are allowed to manage other users and also manage accounts. A normal user can only edit his own profile with some limitations. This module is designed so that the developer self-hosting this application can create multiple accounts for his various clients.

5.4.1 BaseAdmin Presenter

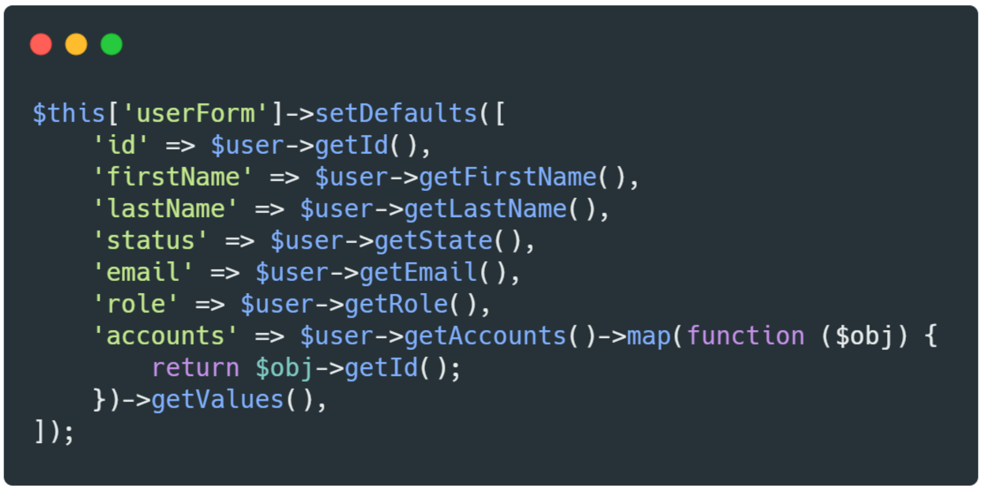
All Admin module presenters extend the `BaseAdminPresenter` class. This class takes care of checking the role of the user via the method `User::isAdmin()` with one exception – if a user is accessing the User form of his own user account, the script lets them through.

5.4.2 Account Presenter

This presenter handles the classic list view and the entity form actions, which work the same as in all other presenters. The interesting part of this presenter is a special handle method which deletes a logo from an account. First, the script finds the account and checks if the current user is authorized to manage the Account entity in question. If this check is successful, the physical file is deleted from the application using the `Utils\FileSystem` class. Then the `logo` attribute of the Account entity is set to null and the entity is persisted.

5.4.3 User Presenter

The User presenter only provides a list view and the entity form without any special methods. If the form method is supplied a user ID, the form action provides the right default values to the form, as seen in Figure 5.4.



```php
$this['userForm']->setDefaults([
    'id' => $user->getId(),
    'firstName' => $user->getFirstName(),
    'lastName' => $user->getLastName(),
    'status' => $user->getState(),
    'email' => $user->getEmail(),
    'role' => $user->getRole(),
    'accounts' => $user->getAccounts()->map(function ($obj) {
        return $obj->getId();
    })->getValues(),
]);
```

Figure 5.4 User Form Default Values

6 UI

The UI section of this application can be found in the `app/UI` directory, and serves the purpose of storing reusable components working across all modules. It contains Latte templates, form factories and all modal windows.

6.1 Latte Components

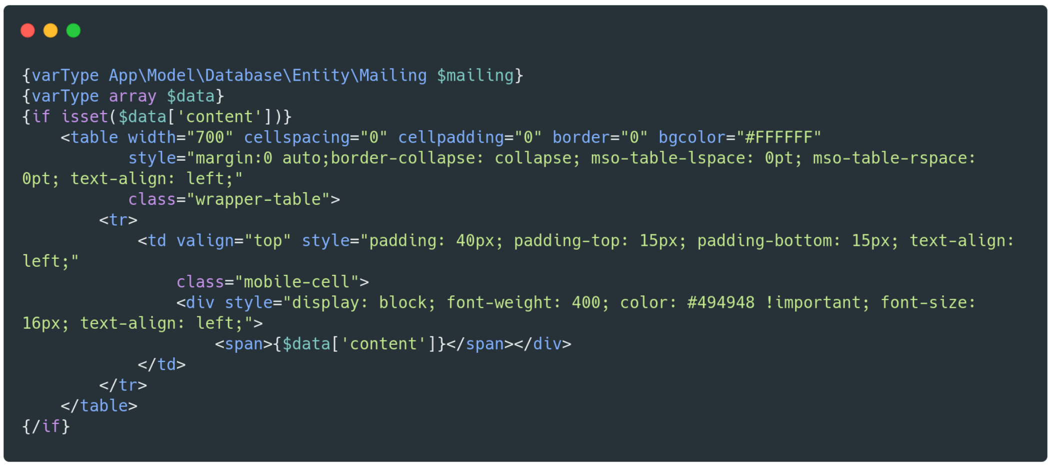
When a Mailing entity is to be sent, it has to be converted into an HTML template in order to be viewed in a MUA. For this reason, every Vue.js editor component has a Latte counterpart in the `app/UI/Components` section. There is also a common layout file inside of which all the components are included. The layout Latte file has important CSS definitions, ensuring the template's styling will work in most e-mail clients.

All Latte components are built as HTML tables because of big differences between rendering engines of MUAs. Although this started to change recently, especially with Gmail's major update last year, some email clients still do not support a lot of HTML and CSS. The most notable: Microsoft's Outlook family of email clients, which use Microsoft Word as their rendering engine. Since Outlook is still hugely popular, email designers have to use tables in some capacity if they want their campaigns to display properly to subscribers. [17] This is clearly visible in a code snippet in Figure 6.1, which is the simplest of all the Latte mailing components used in this application. Notice the table scaffolding with inline styling and non-semantic use of HTML elements – something that would not pass as modern HTML code nowadays.

6.2 Forms

Every form in this application was implemented using the `Application\UI\Form` class – a package which allows developers to define fields, field types, rendering, event handling and validation rules (both client-side and server-side), all within one class. Nette Framework documentation primarily shows how to use Nette Forms in presenters, including their creation and event handling. However, since all forms are essentially Nette components, this application uses factory design pattern[1] (more in Chapter 7.3.4) to create the form separately and then only use the `createComponent<Name>` method in the presenter. This decouples the form from the presenter in a more intuitive way

[1] Factory Design Pattern `https://www.oodesign.com/factory-pattern.html`



```
{varType App\Model\Database\Entity\Mailing $mailing}
{varType array $data}
{if isset($data['content'])}
    <table width="700" cellspacing="0" cellpadding="0" border="0" bgcolor="#FFFFFF"
           style="margin:0 auto;border-collapse: collapse; mso-table-lspace: 0pt; mso-table-rspace:
0pt; text-align: left;"
           class="wrapper-table">
        <tr>
            <td valign="top" style="padding: 40px; padding-top: 15px; padding-bottom: 15px; text-align:
left;"
                class="mobile-cell">
                <div style="display: block; font-weight: 400; color: #494948 !important; font-size:
16px; text-align: left;">
                    <span>{$data['content']}</span></div>
            </td>
        </tr>
    </table>
{/if}
```

Figure 6.1 Latte Template of the Paragraph Component

and also makes it possible for all form classes to reside in the UI section. Because all components are created in separate methods, the code is cleaner and easier to read.

6.2.1 Vulnerability Protection

According to its documentation, Nette Framework puts a great effort into security. Since forms are the most common user input, Nette forms must be impenetrable. In Nette forms, all is maintained dynamically and transparently, nothing has to be set manually. In addition to protecting the forms against attacks targeted at well-known vulnerabilities such as Cross-Site Scripting (XSS, Chapter 3.1.2) and Cross-Site Request Forgery (CSRF, Chapter 3.1.1), it does a lot of small security tasks that the developer no longer has to think about. [18] More information about Nette Framework's security can be found in Chapter 3.

7 TECHNOLOGIES

Undoubtedly, today's web application development is largely based on using the right technologies for the task at hand. These tools help programmers streamline their work without having to reinvent the wheel, so to speak, and they come in two main categories – front-end and back-end.

7.1 Front-end

Starting with front-end technologies, web applications can utilize a plethora of client-side tools that help style the content or make it more dynamic. The core of these technologies is the combination of HTML for markup, CSS for styling and JavaScript for interactivity. To make programming more effective, it is a good idea to use extension languages or frameworks, which offer a superstructure allowing developers to trivially solve everyday tasks that are achieved with difficulty in vanilla versions.

In this thesis, HTML has been streamlined with a template engine which ships with Nette Framework: Latte (Chapter 7.1.1). As for styling, the CSS preprocessor Sass (Chapter 7.1.2) has been used. Lastly, the JavaScript framework Vue.js (Chapter 7.1.3) is utilized to implement the interactive mailing editor.

7.1.1 Latte

This template engine is a great help in separating the controller layer from the view layer – in other words, there is no PHP mixed in HTML files, which makes for uncluttered application structure following the MVC design pattern (more in Chapter 7.3.3). Apart from perfectly integrating into Nette Framework, Latte is secure: it is the first PHP templating engine that introduced context-aware escaping and link checking. [19]

7.1.2 Sass

Sass makes writing CSS easier not only by introducing syntactic sugar, but by allowing the developer to define variables and use them across the stylesheets. This allows for high reusability and scalability of stylesheets – without a preprocessor, CSS is very difficult to maintain.

There are a number of CSS preprocessors on the market, but three are on the top, according to Slant.co. [20] With Sass leading the way, Stylus and Less are not far behind. It could be argued that Sass gained popularity since Bootstrap ships with Sass and includes some interesting Sass maps.

7.1.3 Vue.js

Over the last few years, JavaScript frameworks have been a highly discussed topic in web application development. The market is swarming with different JS frameworks and there are even some frameworks on top of frameworks (Nuxt.js for Vue.js and Next.js for React), so it is easy to get confused with this wide selection. This is not necessarily a bad thing, because improving JavaScript was long overdue. jQuery[1] (2006) was one of the first mainstream JavaScript frameworks, but is now being replaced by newer frameworks or Vanilla JavaScript, currently receiving ECMAScript updates every year, all the while rendering jQuery obsolete and needlessly cumbersome.

One might speculate that the tidal wave of JS frameworks was foretold by Jeff Atwood, one of StackOverflow's founders, who famously blogged this statement:

> Any application that can be written in JavaScript, will eventually be written in JavaScript. [21]

Perhaps the most interesting and important dynamic front-end technology used in this thesis, Vue.js, is the driving force of the mailing editor and all of its components. The Vue paradigm assembles a page out of smaller components which can be freely combined – each component has its own microcosm with own HTML, CSS and JS. This makes it ideal for the mailing system's editor, since the user essentially stacks components after each other. This way, the markup and styling is contained to aid in decoupling of JS code.

The whole editor is a Vue component of its own, providing a list of sub-components a user can add to the editor canvas. This is achieved using the `v-on` directive, which listens to DOM (Document Object Model) events and runs specified JavaScript code when the events are triggered. [22] In this case, each components has its own `v-on:click` directive pointing to a method. The algorithm in these methods is analogous in each sub-component – it consists of saving the index of the latest component added, initializing the new item with respective parameters and committing the item to the Vuex[2]

[1] jQuery `https://jquery.com/`
[2] Vuex `https://vuex.vuejs.org/`

store (more in 7.1.4).

To provide a dynamic preview, the main Vue component iterates over all the components in the Vuex store and calls their template, providing the component's type and key. This is necessary, since there can be multiple components of the same type. Each component is then rendered according to the rules encapsulated in the component's code, which makes for a very clean and scalable JavaScript code. If a developer decides to create new components, e.g. a gallery, it is easy to do so following this paradigm.

7.1.4 Vuex

This application uses Vuex 3 to introduce the state management pattern to Vue.js. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion. [23] This proves useful when changing the state of the editor body or reordering the editor components. For each of these actions, there is a Vuex mutation carrying it out inside the Vuex store.

7.1.5 Bootstrap 5

Bootstrap is an HTML, CSS and JS framework, offering a toolbox of useful libraries that make developing a responsive and semantic front-end code easier. These are some of the reasons to use Bootstrap:

- The web application is abstracted into a set of components and layouts
- Bootstrap consolidates[3] differences caused by distinct rendering engines of browsers
- It provides a grid layout and flex utilities[4], making it easy to define different layouts for each display width
- Bootstrap ships with its own JS functions like collapse[5], modal[6], tooltips[7] etc.

[3] Bootstrap Reboot `https://getbootstrap.com/docs/5.0/content/reboot/`
[4] Bootstrap Flex `https://getbootstrap.com/docs/5.0/utilities/flex/`
[5] Bootstrap Collapse `https://getbootstrap.com/docs/5.0/components/collapse/`
[6] Bootstrap Modals `https://getbootstrap.com/docs/5.0/components/modal/`
[7] Bootstrap Tooltips `https://getbootstrap.com/docs/5.0/components/tooltips/`

Since this project focuses mainly on solid back-end functionality and intuitive UI, a CSS framework like Bootstrap is the optimal choice, making it easy to create modern, responsive and performance-driven templates effectively.

Grid Bootstrap's grid system uses a series of containers, rows, and columns to layout and align content. [24] There are six different grid breakpoint tiers in Bootstrap:

- Extra small (XS)
- Small (SM)
- Medium (MD)
- Large (LG)
- Extra large (XL)
- Extra extra large (XXL)

These breakpoints have predefined widths across which the grid changes, as presented in Table 7.1.

Table 7.1 Bootstrap Breakpoints

	XS	SM	MD	LG	XL	XXL
Display Width	$<$576px	$\geq$576px	$\geq$768px	$\geq$992px	$\geq$1200px	$\geq$1400px
Container	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-	.col-xxl-

Modal In some cases, it is beneficial to use modal windows instead of redirecting the user to a different page. This application uses this approach upon prompting an entity deletion for the user to confirm they actually intend to remove the record. The same principle applies to the import of recipients into groups, displaying the import form inside of a modal window rather than a separate page. The last instance of using a modal is when a user prompts a test sendout of a mailing, so they can specify the desired test e-mail address.

7.2 Back-end

The server-side part of this application is responsible for the functionality of each entity and their interactions. It also provides the database layer, querying the information necessary in each controller, and implements new entry points that interface with the application – such as a CLI.

7.2.1 PHP 8

Web applications can be written in one of many different server-side programming languages, the popular ones currently being PHP[8)], Python[9)], Ruby[10)], C#[11)] and NodeJS[12)]. Each of these languages then offer multiple frameworks extending the usability of that language.

The last years have witnessed a significant overhaul of PHP's internals, bringing it on par with other modern back-end web development languages. The most important changes are these:

- 5.3: Namespace support
- 5.4: Trait support
- 7.0: Scalar type declarations, return type declarations
- 7.1: Nullable types, void functions
- 7.2: Abstract method overriding
- 7.4: Typed properties in classes
- 8.0: Union types, named arguments
- 8.1: Enums

PHP received a lot of censure in its older versions – mainly originating in Wordpress' core, having been written without OOP and never being refactored to adhere to modern

[8)]PHP `https://www.php.net/`
[9)]Python `https://www.python.org/`
[10)]Ruby `https://www.ruby-lang.org/en/`
[11)]C# `https://docs.microsoft.com/en-us/dotnet/csharp/`
[12)]NodeJS `https://nodejs.org/en/`

PHP programming principles. Apart from Wordpress, PHP is used by 79.2 % of all the websites whose server-side programming language is known. [25] It can be stipulated that this created a great deal of legacy code not functioning properly, furthering the disrepute of PHP.

This application is written in PHP 8.0, utilizing strict typing, typed properties in all classes, return type declarations and of course, traits and namespaces. All of this makes PHP a dependable object-oriented web development language, hopefully mending its bad reputation.

7.2.2 Doctrine 2

A big part of web development is modeling, creating and querying the database layer. Since the application's code follows the OOP paradigm closely, it makes sense to also use classes to represent database tables – as separate entities. In Doctrine 2, entities are defined as PHP objects that can be identified over many requests by a unique identifier or primary key. [26] The workflow of a typical ORM is as follows:

1. Writing the entity class, defining its name, attributes and methods
2. Specifying data types for both PHP and Doctrine via annotations
3. Specifying relations for one-to-one, one-to-many, many-to-one and many-to-many related attributes
4. Generating the schema (in SQL) of the actual database via the ORM's schema tool
5. Migrating the generated queries
6. Working with the database via an entity manager and repositories

Working with Doctrine 2 is effective, since the developer only has to write the code once. For example, with Laravel's Eloquent database layer, the developer needs to write a database migration and an Eloquent model, tying the database and application layers together by Laravel's auto-wiring magic. Even then, the developer has to repeat his work, create getter / setter methods and specify type-hints in order to approximate the ease of use of ORM entities. The entities used in this thesis are described in Chapter 4.3.

To easily integrate Doctrine with Nette Framework, this application uses Nettrine package developed by Contributte, a famous group of Czech developers creating open-source code extending Nette Framework.

Traits Some entity attributes would be repeated over multiple classes – e.g. the entity's title, active flag, created/updated time etc. Traits are a different way of achieving composition, which is usually done over inheritance. When there is need of different combinations of features, traits are the ideal solution. It also makes sense from the responsibility point of view, since traits assemble the parts of one class without spreading the responsibility to other classes.

Life-cycle Callbacks Doctrine offers event-based life-cycle callbacks, which are enabled with a simple annotation of the entity class (`HasLifecycleCallbacks`). Once an entity is persisted, Doctrine dispatches a set of events:

- `preRemove` - The `preRemove` event occurs for a given entity before the respective `EntityManager` remove operation for that entity is executed. It is not called for a DQL DELETE statement. [26]
- `postRemove` - The `postRemove` event occurs for an entity after the entity has been deleted. It will be invoked after the database delete operations. It is not called for a DQL `DELETE` statement. [26]
- `prePersist` - The `prePersist` event occurs for a given entity before the respective `EntityManager` persist operation for that entity is executed. It should be noted that this event is only triggered on initial persist of an entity (i.e. it does not trigger on future updates). [26]
- `postPersist` - The `postPersist` event occurs for an entity after the entity has been made persistent. It will be invoked after the database insert operations. Generated primary key values are available in the `postPersist` event. [26]
- `preUpdate` - The `preUpdate` event occurs before the database update operations to entity data. It is not called for a DQL `UPDATE` statement nor when the computed changeset is empty. [26]
- `postUpdate` - The `postUpdate` event occurs after the database update operations to entity data. It is not called for a DQL `UPDATE` statement. [26]

- `postLoad` - The `postLoad` event occurs for an entity after the entity has been loaded into the current `EntityManager` from the database or after the refresh operation has been applied to it. [26]

In this application, all entities have a `createdAt` and `updatedAt` attribute. The `createdAt` setter listens for the `prePersist` event, while `updatedAt` listens for the `preUpdate` event. Once triggered, the attribute is set to the current `DateTime`.

Query Builder While repositories are a great solution for simple fetching of object collections, Doctrine's query builder is optimal for complex queries across multiple joined entities. A `QueryBuilder` provides an API that is designed for conditionally constructing a DQL query in several steps. [26] The DQL is then interpreted into SQL and sent to the database layer, executing the resulting query. With fluent syntax, it is easy to assemble the desired query dynamically. Furthermore, order does not matter, so a `GROUP BY` can be called before a `WHERE` expression – something that would cause an error in pure SQL.

7.2.3 Symfony Console

The command line is a very important interface for this application, since it allows code to run without the constraints of usual browser timeouts and memory limits. Symfony Framework has a very efficient and convenient console component, which can be easily used with Nette via the contributte/console package. An added bonus is that the commands can be fluently tested, asserting the console output or the command's return value.

As expected with CLI commands, arguments and options can be specified, changing the control flow of the command. A good example is the `SendMailingsCommand` class in this application – when started with the option --test, the command does not send any actual e-mails but goes through a dry-run, outputting all e-mail addresses that would get the e-mail in production mode.

7.2.4 Redis

Redis is an open-source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. Redis provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, geospatial indexes, and streams. [27] This application uses a Redis client called Predis[13)] in its Nette-friendly implementation by Contributte[14)]. There is a cache connector for Redis in Nettrine which handles the right configuration of saving temporary files and accessing them promptly.

7.3 Nette

With PHP, it is good practice to use a framework. Frameworks allow fast development and provide well structured, maintainable code base, which makes progressive development much easier, as the application's core is scalable by default. PHP frameworks usually take charge of low-level security, fixing vulnerabilities and security flaws that arise in web development time and again (as explained in detail in Chapter 3). Frameworks also enforce using design patterns, such as factory and MVC (more in Chapters 7.3.3 and 7.3.4), or even the OOP paradigm.

This thesis uses Nette Framework (developed in Czechia), which is essentially a set of decoupled packages working together. In terms of popularity, Nette is certainly not the most used framework on the planet – it is not even included in mainstream surveys, as seen on the graph in Figure 7.1.

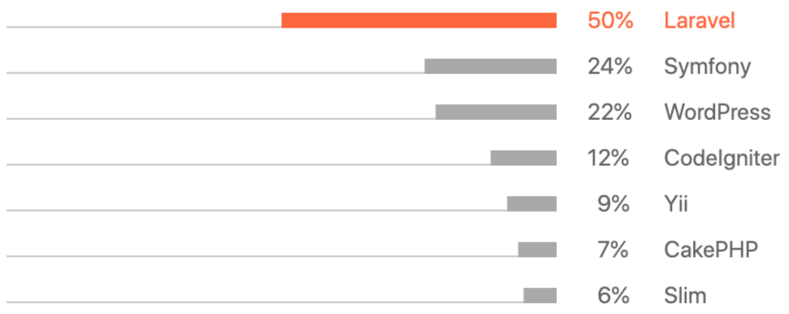


Figure 7.1 Jetbrains Survey of PHP Framework Use

Global polls are steadily dominated by Laravel, closely followed by Symfony. In the local optic, however, Nette Framework is overall more popular than Laravel, as seen in the Czech Google Trends chart in Figure 7.2. Next chapters are going to summarize some interesting features of Nette Framework.

13)Predis `https://github.com/predis/predis`
14)Contributte Redis `https://github.com/contributte/redis`

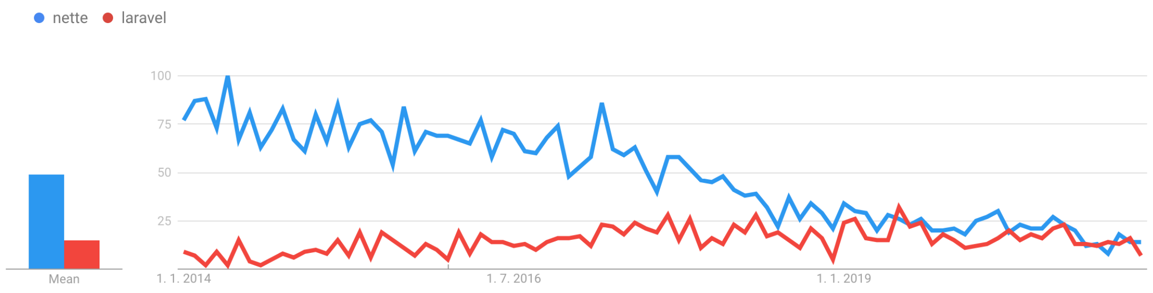


Figure 7.2 Google Trends Chart Comparing Laravel and Nette Popularity in Czechia

7.3.1 Signals

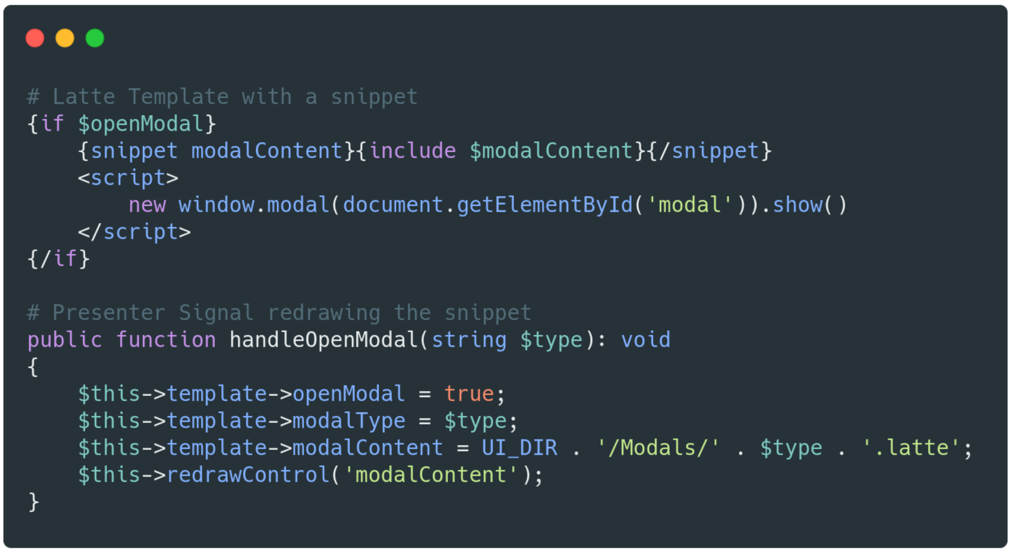
A signal is similar to a normal link in Nette Framework with a small difference – it extends to all classes extending `UI\Control`, offering a more universal way of creating links. A signal is always executed in the context of the current presenter, so it is frequently used in components. A good example could be a `ProductList` component which creates a pagination subcomponent. When a user clicks the next page button, a signal has to be sent to the subcomponent (ideally via an asynchronous call using AJAX), which would be impossible with normal Nette links.

7.3.2 Snippets

Nette provides an interesting interface to deal with asynchronous changes of parts of the web page. Using a pair macro, the developer defines so-called snippets, which represent parts of code that can be redrawn upon request. This could be the counter of products placed in the cart in an e-commerce environment, which needs to change asynchronously once the amount of items in the cart changes. This application uses snippets to open modal windows (which are part of Bootstrap Framework, detailed in Chapter 7.1.5) and provide the right data via back-end. This is achieved with the combination of a snippet and a signal, as presented in Figure 7.3.

7.3.3 MVC Design Pattern

MVC architecture is a modern design pattern of developing applications. It separates the application into three main layers: Model, View and Controller. The Model contains the Business Logic Layer (BLL) that processes the application data and also stores or retrieves data to or from the database. The View (Presentation Layer) displays information to the user. The Controller (Controlling Logic Layer) handles user



```
# Latte Template with a snippet
{if $openModal}
    {snippet modalContent}{include $modalContent}{/snippet}
    <script>
        new window.modal(document.getElementById('modal')).show()
    </script>
{/if}

# Presenter Signal redrawing the snippet
public function handleOpenModal(string $type): void
{
    $this->template->openModal = true;
    $this->template->modalType = $type;
    $this->template->modalContent = UI_DIR . '/Modals/' . $type . '.latte';
    $this->redrawControl('modalContent');
}
```

Figure 7.3 Snippet and Signal Use for Modal Window

interactions and input. [28] A diagram of the MVC pattern is shown in Figure 7.4.

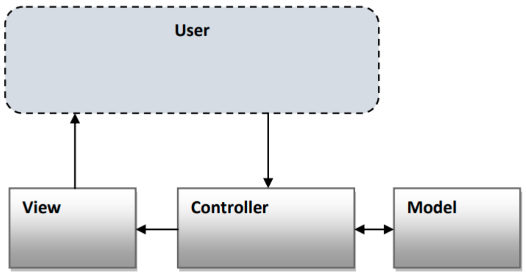


Figure 7.4 MVC Diagram [28]

7.3.4 Factory Design Pattern

Factory is a creational design pattern used to transfer the logic of creating an object into a separate class. This makes sense when creating forms, for example – each factory has a `create()` method and can implement different event handlers, like `onSuccess`, `onError` or `onValidation`. This helps to keep the form logic in one place, making all forms reusable across multiple presenters. When a form is supplied using a factory, the presenter only needs to inject the factory and then use the `createComponent<Name>()` method to call the `create()` method of the injected factory object.

7.4 VCS

Since this application is meant to be open-source and shared with other developers, it is absolutely necessary to use a VCS to enable concurrent work, feature branch development, release tagging and issue tracking. With these mechanisms in place, it is possible to effectively service the package.

7.4.1 Git

In this thesis, the VCS of choice is the industry-standard Git hosted on Github[15]. Git is a reliable technology used by the majority of developers, offering helpful tools like GitFlow[16], which enforce good principles of version control workflow. The comparison of usage of different VCS platforms is shown in Figure 7.5.

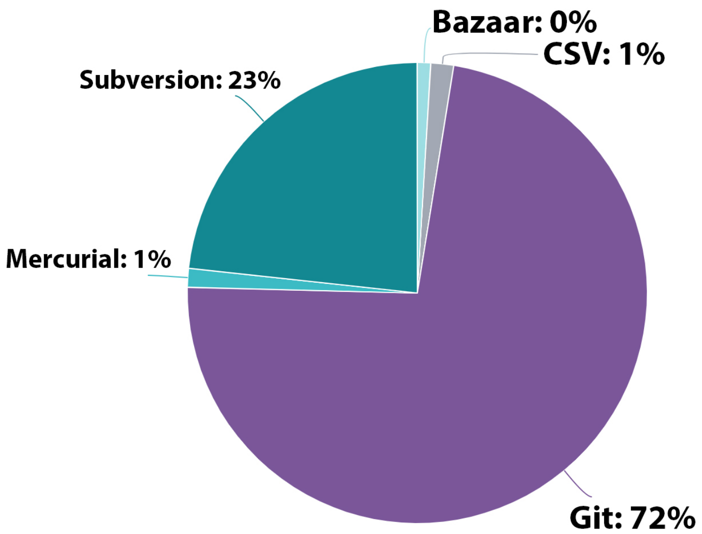


Figure 7.5 OpenHub VCS Use Comparison Pie Chart

[15] https://github.com/jvitasek/nwsltr
[16] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

8 EXTENSIONS

The power of a web application framework lies in its extensibility. It is fairly simple to extend the functions of Nette Framework, since it provides an extensions section in its configuration files. This section is used to register classes extending the class `CompilerExtension`. When registered, these classes are made available in Nette's DI container and can be injected into the application.

8.1 Application Structure

Nette Framework only comes with a hint of how the application should be structured, laying out basic rules on where public files should be stored, which directory serves for application code, or where logs and temporary files will be stored. Apart from that and the default structure for Presenter + Template (Controller + View), there are no best practice instructions put forth. This is why this application follows the structure rules and principles of Webapp Skeleton.

Webapp Skeleton[1)] is one of the libraries developed by Contributte. It essentially shows how to structure a Nette application with focus on PSR-4 compliance, how to separate configuration (Neon[2)]) files and how to change the autowiring of presenters / templates to work in a more concise way.

The important feature of Webapp Skeleton is a trait called `StructuredTemplates`, which overrides the `formatLayoutTemplateFiles` and `formatTemplateFiles` methods of the Presenter class. This allows the presenter and template files to be structured like this: `%presenterDir%/templates/%view%.latte`. This is shown in Figure 8.1, where Front is the module, Editor is the presenter and there are 2 views – default.latte and preview.latte.

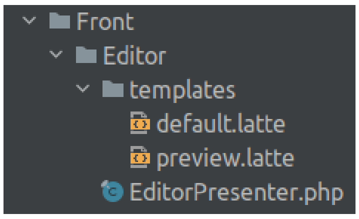


Figure 8.1 Custom Application Structure

1) Webapp Skeleton `https://github.com/contributte/webapp-skeleton`
2) Neon `https://doc.nette.org/en/3.1/neon`

This trait comes from the package Contributte/Application[3] which is implemented in Webapp Skeleton by default, along with some custom presenter response classes and adapters. This structuring would be impossible with the native Nette settings, where the structure for templates is `%moduleDir%/templates/%presenterName%` and `%moduleDir%/presenters/%presenterName%` for presenters.

8.2 Third-party Libraries

nettrine/orm This package[4] integrates Doctrine 2 (more in Chapter 7.2.2) into Nette Framework, allowing easy registration into the Dependency Injection container and effective use across a Nette application.

contributte/console This package[5] integrates Symfony Console (more in Chapter 7.2.3) into Nette Framework, since it is used to execute commands.

ublaboo/api-router As mentioned in Chapter 5.3, this application uses this package[6] to route API requests from front-end (Vue.js) to back-end (Nette Framework).

contributte/forms-bootstrap Since Nette offers automatic rendering of forms and the front-end uses Bootstrap 5, it is important to render the form according to Bootstrap standards. This package[7] achieves that.

phpoffice/phpspreadsheet This package[8] is used when a user imports recipients from an Excel file (mentioned in Chapter 11.1). It parses the file so PHP can use the data.

nettrine/extensions-beberlei This package[9] integrates beberlei/DoctrineExtensions[10] – a library extending Doctrine 2 with MySQL functions like `EXP`, `FLOOR`, `YEAR` etc.

jvitasek/DumpAndDie This package[11] integrates a Laravel dumping function into Tracy (the native debugger in Nette Framework), which is helpful both in development and production environment.

[3] Contributte/Application `https://github.com/contributte/application`
[4] nettrine/orm `https://github.com/nettrine/orm`
[5] contributte/console `https://github.com/contributte/console`
[6] ublaboo/api-router `https://github.com/contributte/api-router`
[7] contributte/forms-bootstrap `https://github.com/contributte/forms-bootstrap`
[8] phpoffice/phpspreadsheet `https://github.com/PHPOffice/PhpSpreadsheet`
[9] nettrine/extensions-beberlei `https://github.com/nettrine/extensions-beberlei`
[10] beberlei/DoctrineExtensions `https://github.com/beberlei/DoctrineExtensions`
[11] jvitasek/dumpanddie `https://github.com/jvitasek/DumpAndDie`

9 STATIC ANALYSIS

While static analysis itself is a sizable topic out of the scope of this thesis, it is important to understand why it is beneficial to use it in PHP applications. Static analysis can detect problems in the code without executing it, which is paramount in interpreted languages like PHP. Compiled languages check each line of code for errors before actually compiling and allowing the developer to run the binary. Static analysis brings PHP closer to this work flow, where the developer can be sure there are no hidden problems that will arise during runtime. This thesis uses 2 static analysis libraries described in Chapters 9.1 and 9.2.

9.1 PHPStan

A very well maintained and popular Czech library, PHPStan[1)] (and its Nette extension[2)]), is a great tool for static analysis in PHP applications. It allows the developer to configure the analysis' scope, level and even exclude some checks. All is set up via a NEON configuration file, native to Nette Framework.

9.2 PHP_CodeSniffer

PHP_CodeSniffer[3)] is a set of two PHP scripts; the main `phpcs` script that tokenizes PHP, JavaScript and CSS files to detect violations of a defined coding standard, and a second `phpcbf` script to automatically correct coding standard violations. [29] However, CodeSniffer is only a tool and requires coding standard (called sniffs) to be provided. This application uses a library called Slevomat Coding Standard[4)] which offers a substantial amount of various sniffs sorted into categories:

- Functional: improving the safety and behaviour of code [30]
- Cleaning: detecting dead code [30]
- Formatting: making the code look consistent [30]

1)PHPStan `https://github.com/phpstan/phpstan`
2)PHPStan Nette `https://github.com/phpstan/phpstan-nette`
3)PHP_CodeSniffer
4)Slevomat Coding Standard `https://github.com/slevomat/coding-standard`

10 AUTOMATED TESTS

To avoid the need to test all the application's functionality after every change or new feature, it is essential to set up and create automated tests. In this thesis, there are two types of automated tests – acceptance tests and unit tests. Each of them serves a different purpose and works in a unique way, providing important information about the system's functionality.

10.1 Acceptance Tests

The first type of automated tests in this application are acceptance tests. They basically simulate a non-technical person testing the application, abstracting actions like clicking a button or filling a form into a high-level domain language. As Dave Farley stated in his interview with InfoQ, acceptance tests are an unalienable part of automated testing:

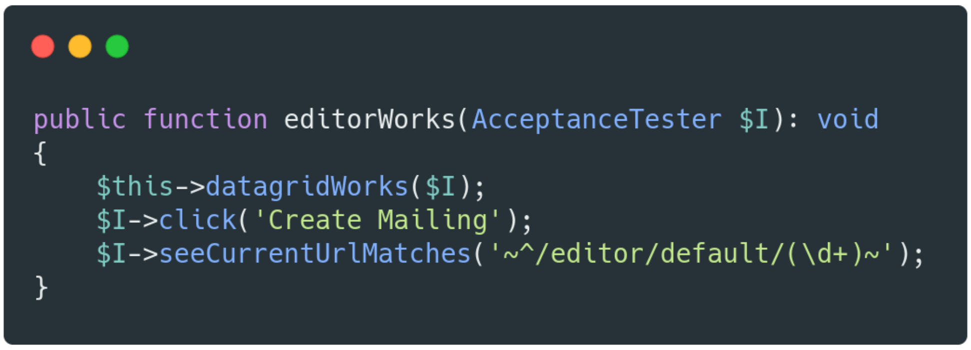
> I see automated acceptance tests as an essential component of a Continuous Delivery style testing strategy. Combined with low-level unit tests, best created as the fruits of Test Driven Development, acceptance tests give us an important and different insight into the behaviour of our systems. [31]

In this thesis, the technology behind automated acceptance tests is Codeception[1]. Tests are represented as a set of user's actions, covering scenarios from a user's perspective. With acceptance tests, the developer can be confident that users, following all the defined scenarios, will not get errors. [32] Codeception provides an ideal platform to create object-oriented acceptance tests with intuitive syntax. It's action API allows to check for elements in the page's body or to compare the current URL against a regular expression, as shown in Figure 10.1.

10.2 Integration Tests

Unlike acceptance tests, integration tests focus on verifying that the application's modules work together as expected. For this test section, there is no more browser simulation, so the tests need an actual database configuration and the right PHP .ini settings

[1] Codeception `https://codeception.com/`



```php
public function editorWorks(AcceptanceTester $I): void
{
    $this->datagridWorks($I);
    $I->click('Create Mailing');
    $I->seeCurrentUrlMatches('~^/editor/default/(\d+)~');
}
```

Figure 10.1 Codeception Acceptance Test of the Mailing Editor

to be able to execute the actions inside the test cases. In this application, both integration and unit tests are run in Nette Tester[2]. This allows tests to seamlessly integrate into Nette Framework without any setup difficulties.

10.2.1 Database

For this mailing system, the integration tests entail two parts – Database and Latte. For the system to function, the right database configuration is essential, and since the database is handled through Doctrine ORM (Chapter 7.2.2), the tests can validate entity mappings. Once Doctrine returns any failed mapping, the test case prints out the problem so it can be fixed. If no problems are found, the test succeeds. Another important thing to test in regards to the database is the `TRepository` trait – since it offers an easy way to access repositories (Chapter 4.4), it should define getter methods for all defined repositories. If it does not, the test fails.

10.2.2 Latte

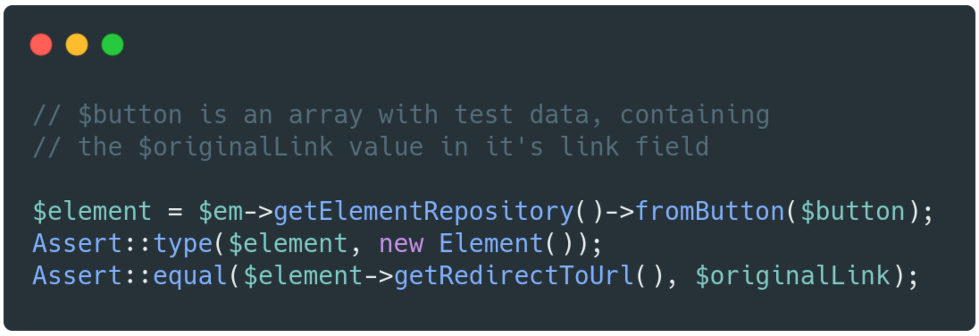
Since all HTML templates in this application are compiled from Latte templates, it is important to check if all templates can actually be compiled. For example, Latte files could contain syntax errors like an opened macro without closing tags, which would cause the template to throw an exception instead of compiling properly. In the Latte integration test, the script finds all Latte files and tries to compile each of them. If

[2]Nette Tester `https://tester.nette.org/en/`

any exception is caught, the test will fail, showing the path to the last tested Latte template.

10.3 Unit Tests

One of the benefits of following the OOP paradigm is that code is decoupled and easy to test with unit tests, since all functionality is exploded into separate logical units. In this application, unit tests are implemented with the help of Nette Tester (as mentioned in Chapter 10.2) and are used to verify that crucial parts of the system work as expected. One of these cases could be the test checking if the Element Repository method `fromButton(<array>)` (detailed in Chapter 4.4.1) correctly creates an Element object and returns it. This test is shown in Figure 10.2 in an abridged form.



```php
// $button is an array with test data, containing
// the $originalLink value in it's link field

$element = $em->getElementRepository()->fromButton($button);
Assert::type($element, new Element());
Assert::equal($element->getRedirectToUrl(), $originalLink);
```

Figure 10.2 Unit Test to Check fromButton(<array>) Functionality

11 USER TESTS

It is fairly obvious that no number of automated tests can replace a real person using the application and providing relevant feedback. That is why the beta version of this application was sent to multiple users with different levels of technical skills. This is the set of test users:

- User A: a PHP developer with years of experience
- User B: a frequent web application user
- User C: a typical inexperienced internet user

Their task was the same – go through the whole system, get familiar with it, create a mailing template and plan a sendout, all the while making notes of things that could be done better or parts of the system that were unclear to them. The next chapter lists the most important feedback received from user testing.

11.1 Add Import File Sample

The Recipient Group datagrid offers an import function which allows users to upload recipient data in an Excel2007 file, create recipients and add them into a selected group. User A correctly raised a question asking about the format of the .xlsx file: how should it be structured and in which column should the e-mails reside? For these purposes, a sample file was created and linked to in the import modal window. This way, the user can download the sample file and add their own data according to the ready-to-use structure.

11.2 Add Test Sendout Function

An interesting feature was suggested by test user B, who is a frequent user of Mailchimp, professionally designing mailings and managing sendouts for multiple companies. He mentioned the absence of the test sendout feature that mainstream platforms offer for quickly checking if the template displays correctly in their MUA. In light of this feedback, this application adopted the responsive preview functionality (more in Chapter 2.1.9) and now offers it as well.

11.3 Save Mailing When Clicking on Preview

Test user C provided a valuable feedback regarding the application's mailing editor. As an inexperienced user, they clicked on preview, expecting the editor to automatically save the data, and consequently lost all their progress. The editor was updated so that if a user decides to preview the built template during the creation process, it saves all progress automatically.

12 PUBLISHING

As mentioned in the assignment of this thesis and throughout its content, the final application is to be made public as open-source software. Even though this posed some security challenges dealt with in Chapter 3, it is important to give to the community and share knowledge with others, especially in a relatively small society surrounding Nette Framework.

12.1 Git / Github

This application was developed using Git and is hosted on Github since the start, so it makes sense to use Github as a publishing tool. Also, Github is undoubtedly the most used open-source code hosting platform, flaunting more than 56 million registered developers and 60 million repositories added last year. [33] In the future, a developer can create a new feature or a bug-fix in a forked repository, create a pull request to the main repository and then have his changes merged into the production branch of the package.

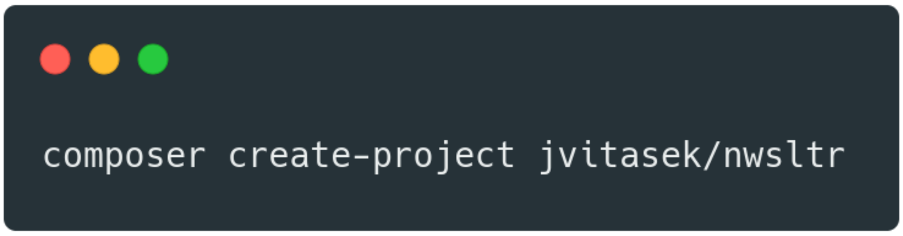
12.2 Composer / Packagist

Composer is a dependency management tool for PHP projects and is one of the most important parts of this application, since it is used to fetch all necessary third-party software in the correct version decided by the server configuration. Apart from easily fetching libraries to different applications, Composer can also be used to fetch whole projects – this can be achieved in the composer.json file via the type attribute. These are the supported types:

- `library`: This is the default. It will copy the files to the `vendor` directory. [34]
- `project`: This denotes a project rather larger than a library. For example application shells like the Symfony standard edition or full fledged applications distributed as packages. [34]
- `metapackage`: An empty package that contains requirements and will trigger their installation, but contains no files and will not write anything to the filesystem. [34]

- `composer-plugin`: A package of this type may provide an installer for other packages that have a custom type. [34]

As can be derived from the list above, the appropriate type for this application is `project`. When published in a Composer repository like Packagist[1)], Composer will allow users to download the whole project with this command shown in Figure 12.1.



```
composer create-project jvitasek/nwsltr
```

Figure 12.1 Composer Command to Fetch the Application

[1)]Packagist `https://packagist.org/`

CONCLUSION

This thesis has tried to show and compare the key parameters of mainstream commercial mailing platforms, while emphasizing best practices of both various user interfaces and functionality features. This entailed carrying out a detailed analysis of existing solutions to find the optimal ways to solve typical mailing system problems – functionality like e-mail opening detection, sendout algorithm, mailing template editor and so on. An important part of this analysis was also selecting the right technology to achieve the intuitiveness and fluency of the user interface, like the combination of Vue.js and Bootstrap.

The findings of this study have been largely implemented into the resulting open-source web application, which has been published on the aforementioned platforms Github and Packagist. The application meets all the important requirements arrived upon in the theoretical part of this thesis, allowing the system to be used by inexperienced users without the need of extensive support. Overall, by using Nette Framework and focusing on the security of the application, it could also prove useful to many developers in need of a ready-made extensible self-hosted mailing solution.

There are many highlights that should be mentioned in closing. At the time of writing this, PHP 8 is still a fairly new release and there are not many open-source applications adopting its new features so early. Using this bleeding edge PHP version during development, it is going to be easier to support and maintain the library for the years to come. Furthermore, the whole code is strictly typed (using the `declare(strict_types=1)` directive) and focused on following principles of OOP, which makes the system's code modern and ideal for debugging, extending and testing. On a related note, the application comes with prepared acceptance tests, feature tests, unit tests and static analysis scaffolding, which can be built upon in the future to increase code coverage.

Future research and development could examine more features detailed in this thesis – e.g. bounce management, predesigned templates, various layouts and so on. The future possibility of discovering a new method of e-mail opening detection warrants further investigation, which could lead to much more effective ways of handling statistics collection and developing new components. In the meantime, this application can serve as a great open-source alternative of Mailchimp and other commercial platforms.

REFERENCES

[1] *Datanyze* [online]. [visited 2021-03-20]. Retrieved from: https://www.datanyze.com/market-share/email-marketing--13

[2] *The Best Email Marketing Software for 2021* [online]. [visited 2021-02-26]. Retrieved from: https://www.pcmag.com/picks/the-best-email-marketing-software

[3] *About Mailchimp* [online]. [visited 2021-03-20]. Retrieved from: https://mailchimp.com/about/

[4] *TechCrunch* [online]. [visited 2021-03-21]. Retrieved from: https://tcrn.ch/3jeuyeR

[5] *Ecomail* [online]. [visited 2021-03-22]. Retrieved from: https://www.ecomail.cz/about

[6] *listmonk* [online]. [visited 2021-03-23]. Retrieved from: https://listmonk.app

[7] *Merriam-Webster Dictionary* [online]. [visited 2021-03-23]. Retrieved from: https://www.merriam-webster.com/dictionary/best%20practice

[8] *Merriam-Webster Dictionary* [online]. [visited 2021-03-27]. Retrieved from: https://www.merriam-webster.com/dictionary/user%20interface

[9] *CampaignMonitor* [online]. [visited 2021-03-18]. Retrieved from: https://www.campaignmonitor.com/blog/email-marketing/7-stats-that-will-make-you-rethink-mobile-email/

[10] *Mailchimp Knowledge Base* [online]. [visited 2021-03-18]. Retrieved from: https://mailchimp.com/help/about-bounces/

[11] *SNYDER, Chris, Thomas MYER a Michael SOUTHWELL.* Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses. 2010. ?: Springer-Verlag, 2010. ISBN 1430233184.

[12] *Doctrine 2 Documentation - Security* [online]. [visited 2021-03-27]. Retrieved from: https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/security.html

[13] *How Secure is BCRYPT?* [online]. [visited 2021-03-28]. Retrieved from: https://synkre.com/how-secure-is-bcrypt/

[14] *What is GDPR?* [online]. [visited 2021-04-18]. Retrieved from: `https://gdpr.eu/what-is-gdpr/`

[15] *Techterms* [online]. [visited 2021-03-24]. Retrieved from: `https://techterms.com/definition/mvc`

[16] *Nette Documentation - Presenters* [online]. [visited 2021-04-03]. Retrieved from: `https://doc.nette.org/en/3.1/presenters`

[17] *HTML Tables in Email: What could possibly go wrong?* [online]. [visited 2021-03-26]. Retrieved from: `https://www.litmus.com/blog/html-tables-in-email-what-could-possibly-go-wrong/`

[18] *Nette Documentation - Forms* [online]. [visited 2021-03-27]. Retrieved from: `https://doc.nette.org/en/3.1/forms`

[19] *Latte Documentation* [online]. [visited 2021-01-05]. Retrieved from: `https://latte.nette.org/en/guide`

[20] *15 Best CSS Preprocessors* [online]. [visited 2021-04-05]. Retrieved from: `https://www.slant.co/topics/217/~best-css-preprocessors-postprocessors`

[21] *The Principle of Least Power* [online]. [visited 2021-01-22]. Retrieved from: `https://blog.codinghorror.com/the-principle-of-least-power/`

[22] *Vue.js Website* [online]. [visited 2021-02-03]. Retrieved from: `https://vuejs.org/v2/guide/events.html`

[23] *Vuex 3 Documentation* [online]. [visited 2021-02-03]. Retrieved from: `https://vuex.vuejs.org/`

[24] *Bootstrap Grid Documentation* [online]. [visited 2021-03-27]. Retrieved from: `https://getbootstrap.com/docs/5.0/layout/grid/`

[25] *Usage statistics of PHP for websites* [online]. [visited 2021-03-05]. Retrieved from: `https://w3techs.com/technologies/details/pl-php`

[26] *Doctrine 2 Documentation* [online]. [visited 2021-03-12]. Retrieved from: `https://www.doctrine-project.org/projects/doctrine-orm`

[27] *Nette Documentation - Presenters* [online]. [visited 2021-04-03]. Retrieved from: `https://redis.io/`

[28] NARESH THAKUR, Ram a Dr. U.S. PANDEY. *A Study Focused on Web Application Development using MVC Design Pattern.* International Research Journal of Engineering and Technology. 2019, 2019(06/08), 7. ISSN 2395-0056.

[29] *PHP Codesnifer* [online]. [visited 2021-04-04]. Retrieved from: `https://github.com/squizlabs/PHP_CodeSniffer`

[30] *Slevomat Coding Standard* [online]. [visited 2021-04-04]. Retrieved from: `https://github.com/slevomat/coding-standard`

[31] *Dave Farley* [online]. [visited 2021-03-29]. Retrieved from: `https://www.infoq.com/news/2017/04/acceptance-testing-delivery/`

[32] *Codeception – PHP Testing Framework* [online]. [visited 2021-03-28]. Retrieved from: `https://codeception.com/`

[33] *Github – The State of the Octoverse* [online]. [visited 2021-04-01]. Retrieved from: `https://octoverse.github.com/`

[34] *The composer.json Scheme* [online]. [visited 2021-04-01]. Retrieved from: `https://getcomposer.org/doc/04-schema.md#type`

[35] *Nette Documentation - Vulnerability Protection* [online]. [visited 2021-03-27]. Retrieved from: `https://doc.nette.org/en/3.1/vulnerability-protection`

LIST OF ABBREVIATIONS

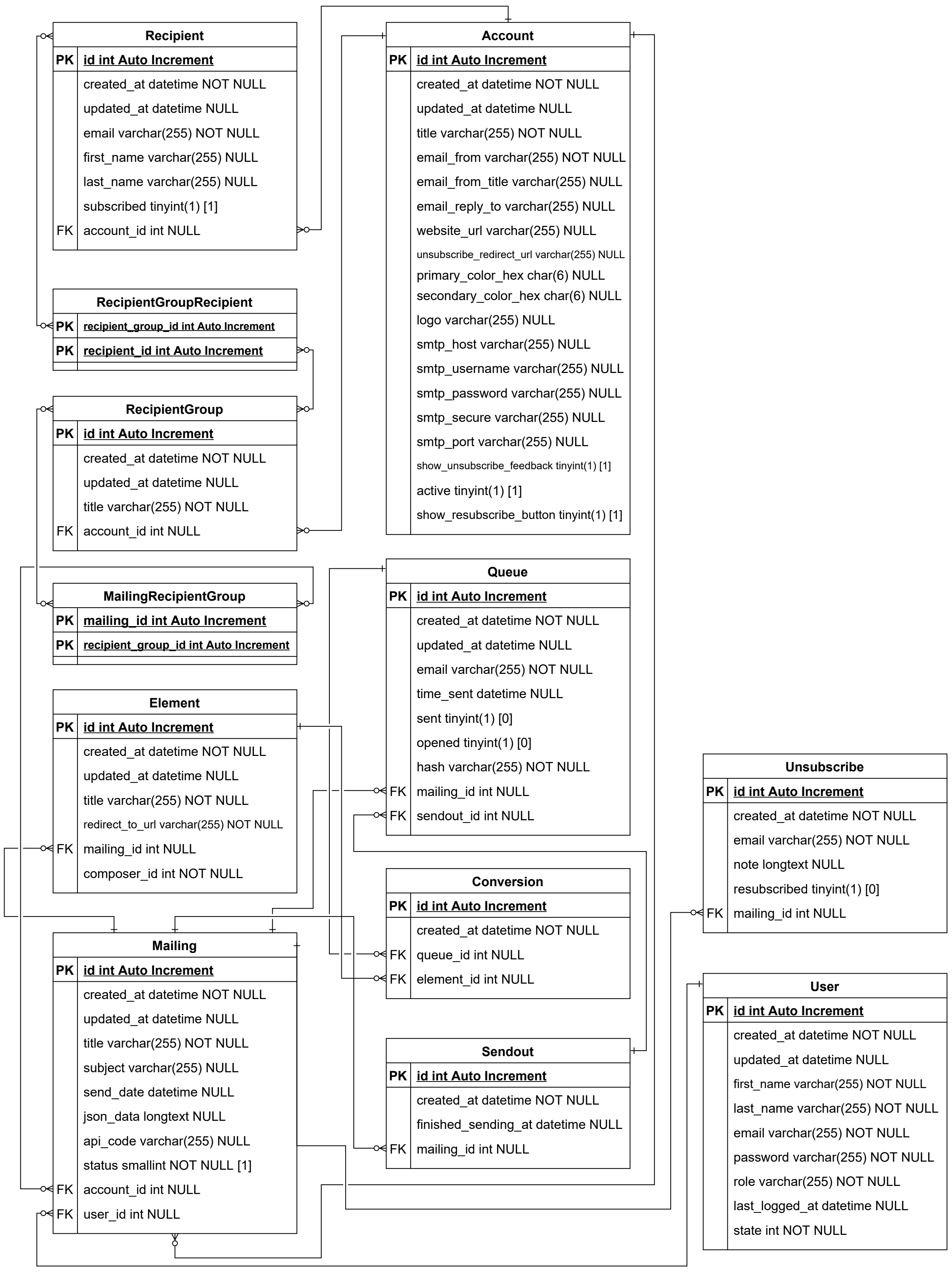
VCS	Version Control System
ORM	Object-relational Mapper
UML	Unified Modeling Language
SMTP	Simple Mail Transfer Protocol
CLI	Command Line Interface
CI	Continuous Integration
CD	Continuous Development
CC	Constant Contact
ORM	Object-relational Mapping
DQL	Doctrine Query Language
DAL	Data Access Layer
DAO	Data Access Objects
MDN	Message Disposition Notification
MUA	Mail User Agent
ISP	Internet Service Provider
SMB	Small and Midsize Business
CRM	Customer Relationship Management
CSE	Comparison Shopping Engines
WYSIWYG	What You See Is What You Get
ERD	Entity-Relationship Diagram
DRY	Don't Repeat Yourself
DI	Dependency Injection
BREAD	Browse Read Edit Add Delete
DAVE	Delete Add View Edit
CRAP	Create Replicate Append Process
AI	Artificial Intelligence
UX	User Experience
MIME	Multipurpose Internet Mail Extensions
PHP	Hypertext Preprocessor
OOP	Object-oriented Programming
AJAX	Asynchronous JavaScript and XML
BLL	Business Logic Layer
MVC	Model-View-Controller
DOM	Document Object Model
GDPR	General Data Protection Regulation
EU	European Union

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

APPENDIX A I. ER DIAGRAM



	Recipient
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	email varchar(255) NOT NULL
	first_name varchar(255) NULL
	last_name varchar(255) NULL
	subscribed tinyint(1) [1]
FK	account_id int NULL

	RecipientGroupRecipient
PK	recipient_group_id int Auto Increment
PK	recipient_id int Auto Increment

	RecipientGroup
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	title varchar(255) NOT NULL
FK	account_id int NULL

	MailingRecipientGroup
PK	mailing_id int Auto Increment
PK	recipient_group_id int Auto Increment

	Element
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	title varchar(255) NOT NULL
	redirect_to_url varchar(255) NOT NULL
FK	mailing_id int NULL
	composer_id int NOT NULL

	Mailing
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	title varchar(255) NOT NULL
	subject varchar(255) NULL
	send_date datetime NULL
	json_data longtext NULL
	api_code varchar(255) NULL
	status smallint NOT NULL [1]
FK	account_id int NULL
FK	user_id int NULL

	Account
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	title varchar(255) NOT NULL
	email_from varchar(255) NOT NULL
	email_from_title varchar(255) NULL
	email_reply_to varchar(255) NULL
	website_url varchar(255) NULL
	unsubscribe_redirect_url varchar(255) NULL
	primary_color_hex char(6) NULL
	secondary_color_hex char(6) NULL
	logo varchar(255) NULL
	smtp_host varchar(255) NULL
	smtp_username varchar(255) NULL
	smtp_password varchar(255) NULL
	smtp_secure varchar(255) NULL
	smtp_port varchar(255) NULL
	show_unsubscribe_feedback tinyint(1) [1]
	active tinyint(1) [1]
	show_resubscribe_button tinyint(1) [1]

	Queue
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	email varchar(255) NOT NULL
	time_sent datetime NULL
	sent tinyint(1) [0]
	opened tinyint(1) [0]
	hash varchar(255) NOT NULL
FK	mailing_id int NULL
FK	sendout_id int NULL

	Conversion
PK	id int Auto Increment
	created_at datetime NOT NULL
FK	queue_id int NULL
FK	element_id int NULL

	Sendout
PK	id int Auto Increment
	created_at datetime NOT NULL
	finished_sending_at datetime NULL
FK	mailing_id int NULL

	Unsubscribe
PK	id int Auto Increment
	created_at datetime NOT NULL
	email varchar(255) NOT NULL
	note longtext NULL
	resubscribed tinyint(1) [0]
FK	mailing_id int NULL

	User
PK	id int Auto Increment
	created_at datetime NOT NULL
	updated_at datetime NULL
	first_name varchar(255) NOT NULL
	last_name varchar(255) NOT NULL
	email varchar(255) NOT NULL
	password varchar(255) NOT NULL
	role varchar(255) NOT NULL
	last_logged_at datetime NULL
	state int NOT NULL